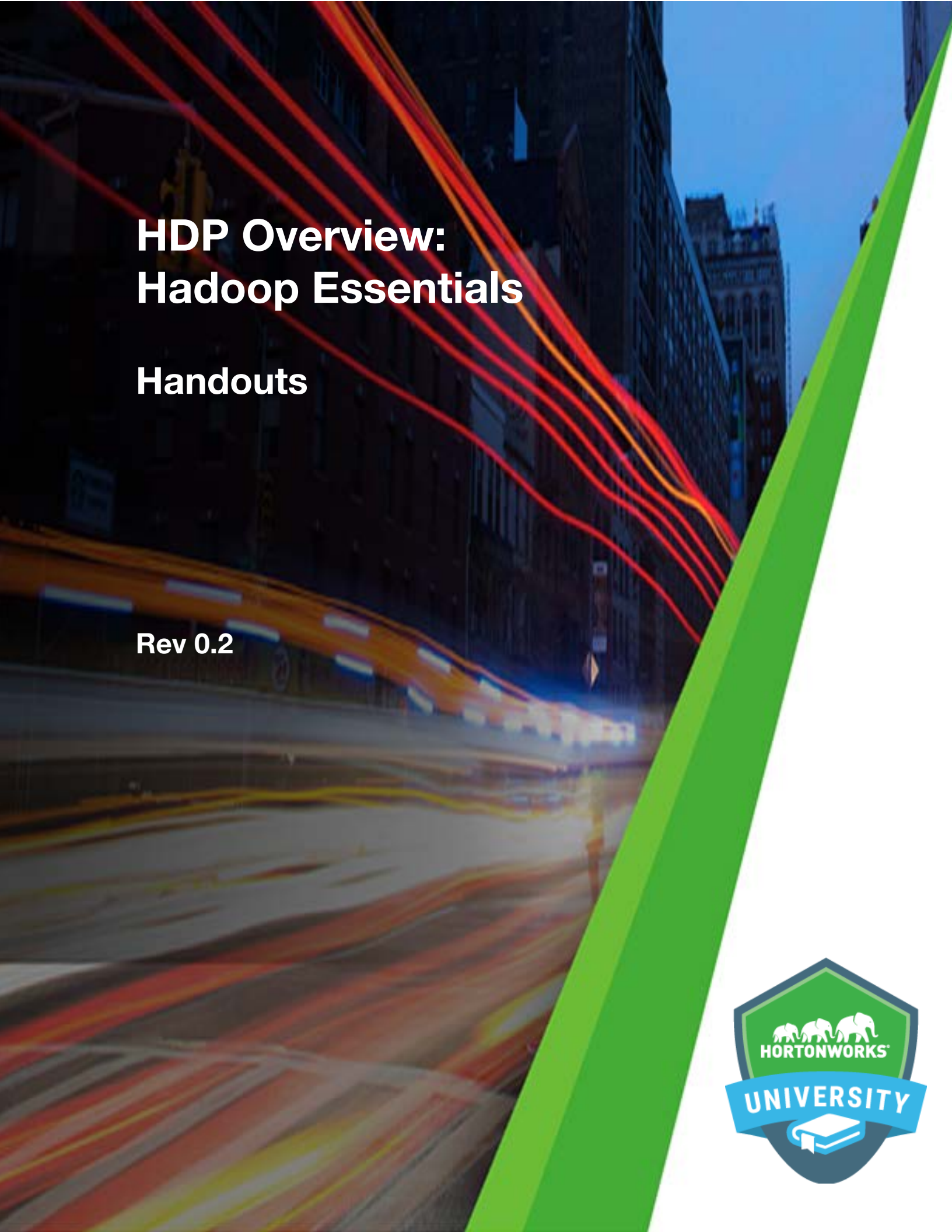


HDP Overview: Hadoop Essentials

Handouts

Rev 0.2

HORTONWORKS UNIVERSITY

Become a ***Hortonworks Certified Professional*** and establish your credentials:

• HDP Certified Developer: for Hadoop developers using frameworks like Pig, Hive, Sqoop and Flume.

• HDP Certified Administrator: for Hadoop administrators who deploy and manage Hadoop clusters.

• HDP Certified Developer: Java: for Hadoop developers who design, develop and architect Hadoop-based solutions written in the Java programming language.

• HDP Certified Developer: Spark: for Hadoop developers who write and deploy applications for the Spark framework.

How to Register: Visit www.examslocal.com and search for “Hortonworks” to register for an exam. The cost of each exam is $250 USD, and you can take the exam anytime, anywhere using your own computer. For more details, including a list of exam objectives and instructions on how to attempt our practice exams, visit http://hortonworks.com/training/certification/

Earn Digital Badges: Hortonworks Certified Professionals receive a digital badge for each certification earned. Display your badges proudly on your résumé, LinkedIn profile, email signature, etc.

Self Paced Learning Library

On Demand Learning

Hortonworks University Self-Paced Learning Library is an on-demand dynamic repository of content that is accessed using a Hortonworks University account. Learners can view lessons anywhere, at any time, and complete lessons at their own pace. Lessons can be stopped and started, as needed, and completion is tracked via the Hortonworks University Learning Management System.

Hortonworks University courses are designed and developed by Hadoop experts and provide an immersive and valuable real world experience. In our scenario-based training courses, we offer unmatched depth and expertise. We prepare you to be an expert with highly valued, practical skills and prepare you to successfully complete Hortonworks Technical Certifications.

Target Audience: Hortonworks University Self-Paced Learning Library is designed for those new to Hadoop, as well as architects, developers, analysts, data scientists, and IT decision makers. It is essentially for anyone who desires to learn more about Apache Hadoop and the Hortonworks Data Platform.

Duration: Access to the Hortonworks University Self-Paced Learning Library is provided for a 12-month period per individual named user. The subscription includes access to over 400 hours of learning lessons.

The online library accelerates time to Hadoop competency. In addition, the content is constantly being expanded with new material, on an ongoing basis.

Visit: http://hortonworks.com/training/class/hortonworks-university-self-paced-learning-library/

Course Agenda

Lecture/Discussion	Demonstration
The Case for Hadoop	
Ecosystem Components and Integrations	*Operational Overview with Ambari*
HDFS Architecture and Features	*Loading Data into HDFS*
Ingesting Data into Hadoop	*Streaming Data into HDFS (time permitting)*
Parallel Processing Fundamentals	*Processing with MapReduce (time permitting)*
Popular Data Transformation and Processing Engines	
Apache Hive	*Data Manipulation with Hive*
Apache Pig	*Risk Analysis with Pig*
Apache Spark	*Risk Analysis with Spark and Zeppelin*
YARN Architecture and Features	
Backup and Recovery	*Data Backup with Falcon (time permitting)*
Hadoop Security	*Securing Hive with Ranger (time permitting)*

Introductions

- Your name
- Your job role and responsibilities
- Your Big Data and/or Hadoop experience, if any
- Your expectations for this course

Class Logistics

- Stopping time for the day
- Breaks and lunch
- Facility information (if applicable)
  - Exits, restrooms, break room…
- Courseware
  - Do you have it?
  - Where do you get it?
- Technical information
- Wireless access, cloud access, virtual machine information…

Lesson Objectives

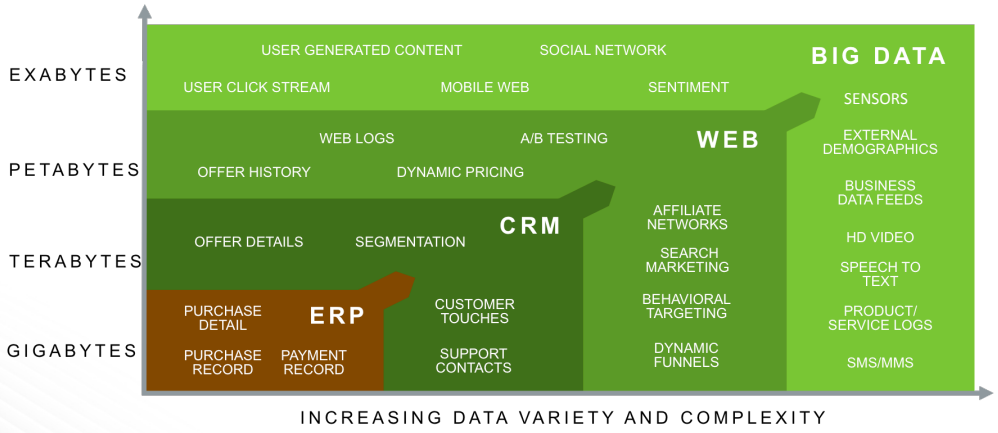
After completing this lesson, students should be able to:

- Describe data trends of volume, velocity & variety
  - Technology threats and opportunities
- List popular use cases for Hadoop
- Discuss the importance of Open Enterprise Hadoop
  - Open
  - Central
  - Interoperable
  - Ready
- Give an overview of Connected Data Platforms powered by Hadoop

Objectives

- Data Trends

The 3 V's of DATA are Driving Apache Hadoop



What Makes Data Big Data?

- The term *Big Data* comes from the computational sciences
- It is used to describe scenarios where the volume and types of data overwhelm the tools to store and process it

Variety	Unstructured and semi-structured data is becoming as strategic as the traditional structured data.
Volume	Data coming in from new sources as well as increased regulation in multiple areas means storing more data for longer periods of time.
Velocity	Machine data, as well as data coming from new sources, is being ingested at speeds not even imagined a few years ago.

VOLUME *Velocity* Variety

Volume

Volume refers to the amount of data being generated.

- Gigabytes, terabytes, petabytes, exabytes, zettabytes …
- Many factors contribute to the increase in data volume, including:
  - Transaction-based data stored through the years
  - Unstructured data streaming in from social media
  - Increasing amounts of sensor and machine-to-machine data being collected
- Problems related to volume include:
  - Storage costs
  - Determining relevance within large data volumes
  - How to analyze data quickly to maximize business value

Velocity

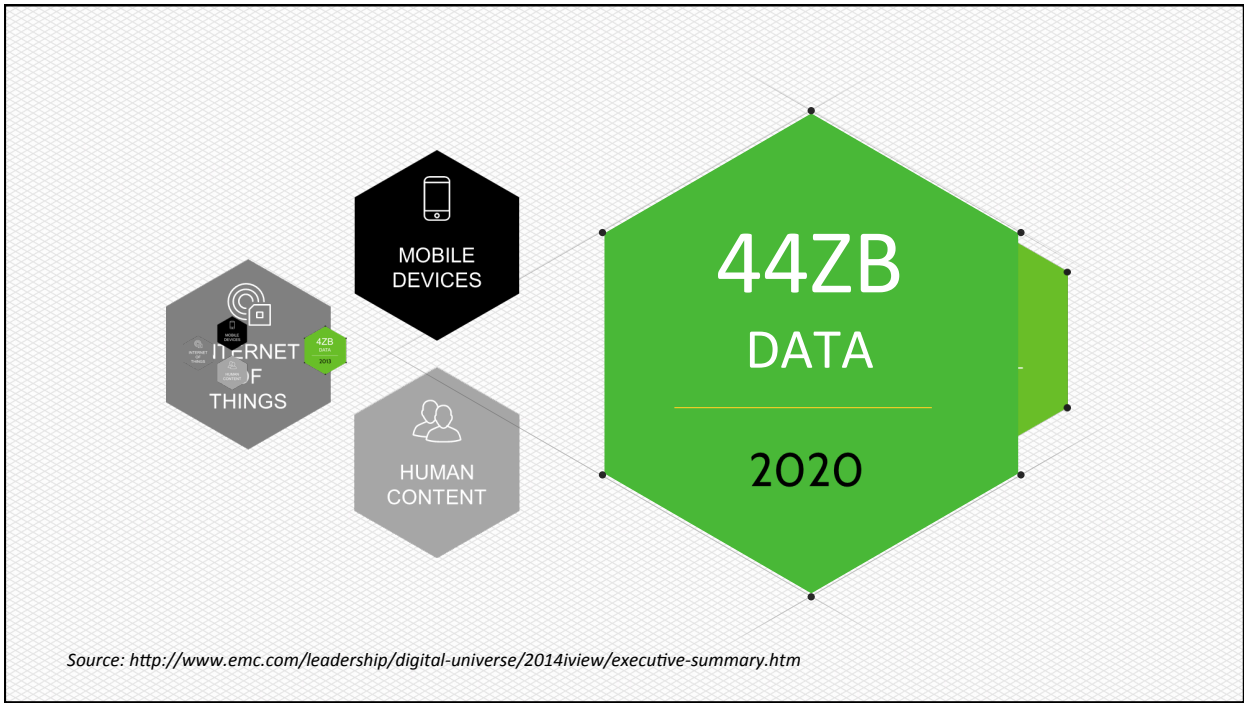
Velocity refers the rate at which new data is generated.

- Megabytes per second, gigabytes per second…
- Data is streaming in at unprecedented speed and must be dealt with in a timely manner in order to extract the maximum value
  - Sources include logs, social media, RFID tags, sensors, and smart metering
- Problems related to velocity include:
  - Reacting quickly enough to benefit from the data
  - Inconsistent data flows with periodic peaks
    - Daily
    - Seasonal
    - Event-triggered

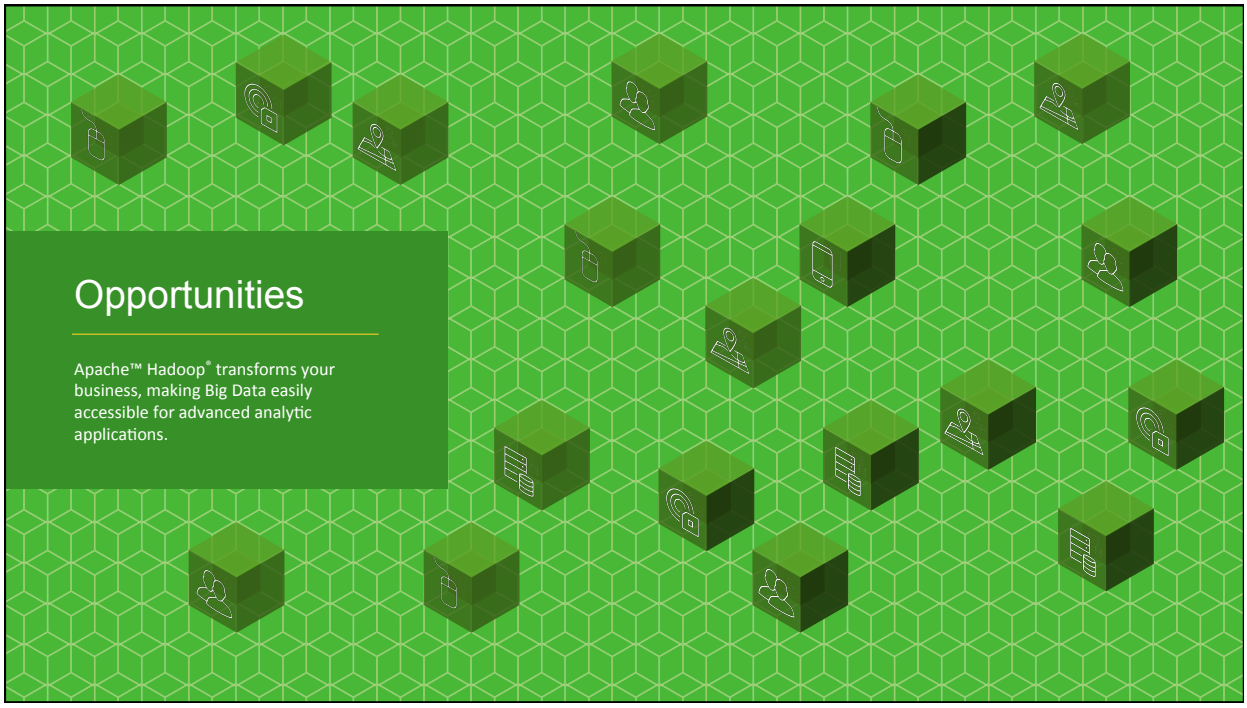
Variety

Variety refers to the number of types of data being generated.

- Varieties of data include:
  - Structured data in traditional databases
  - Semi-structured data like XML or JSON files
  - Unstructured text documents, email, video, audio, stock ticker data, and financial transactions
- Problems related to variety include:
  - How to gather, link, match, cleanse, and transform data across systems
  - How to connect and correlate data relationships and hierarchies to extract business value



Source: http://www.emc.com/leadership/digital-universe/2014iview/executive-summary.htm



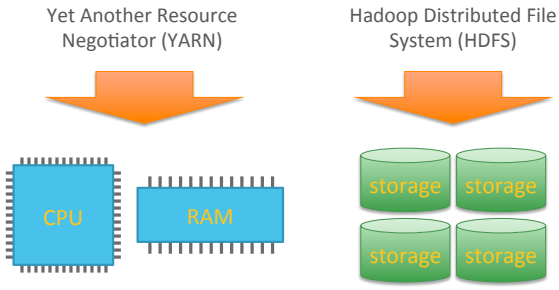
What is Apache Hadoop?

The Apache Hadoop project describes the technology as a software framework that:

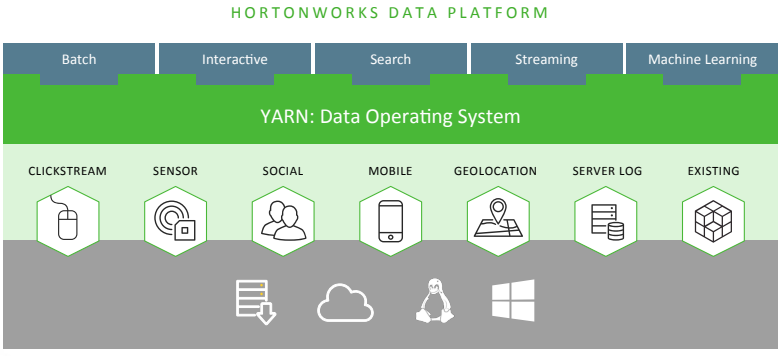
- Allows for the distributed processing of large data sets across clusters of computers using simple programming models
- Is designed to scale up from single servers to thousands of machines, each offering local computation and storage
- Does not rely on hardware to deliver high-availability, but rather the library itself is designed to detect and handle failures at the application layer
- Delivers a highly-available service on top of a cluster of computers, each of which may be prone to failures

Source: http://hadoop.apache.org

Hadoop Core = Storage + Compute

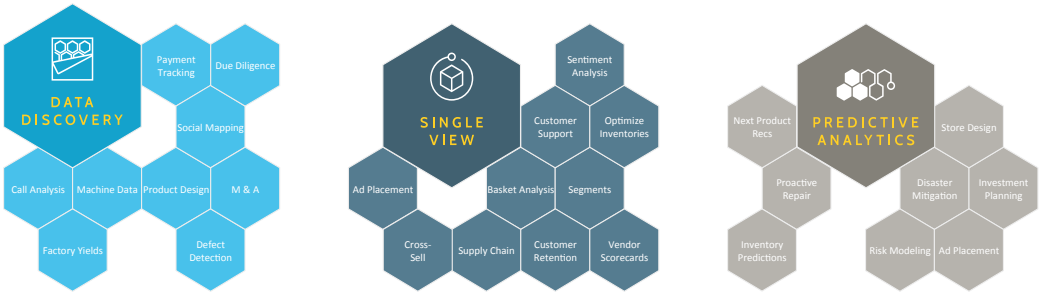


Hortonworks Delivers Open Enterprise Hadoop



Objectives

- Data Trends
- Popular Use Cases for Hadoop



BUSINESS OUTCOMES

Business executives are driving transformational outcomes with next-generation applications that empower new uses of Big Data including: data discovery,
a single view of the customer and predictive analytics.

COST SAVINGS

IT executives are delivering substantial reductions in operating costs by modernizing their data architectures with Open Enterprise Hadoop. These cost saving innovations include active archive of cold data, offloading ETL processes and enriching existing data.

CUSTOMER JOURNEY

Hortonworks® customers leverage our technology to transform their businesses, either by achieving new business objectives or by reducing costs. The journey typically involves both of those goals in combination, across many use cases.

New Analytic Applications for New Types of Data

Financial Services

- New Account Risk Screens
- Fraud Prevention
- Trading Risk
- Maximize Deposit Spread
- Insurance Underwriting
- Accelerate Loan Processing

Retail

- 360° View of the Customer
- Analyze Brand Sentiment
- Localized, Personalized Promotions
- Website Optimization
- Optimal Store Layout

Telecom

- Call Detail Records (CDRs)
- Infrastructure Investment
- Next Product to Buy (NPTB)
- Real-time Bandwidth Allocation
- New Product Development

Manufacturing

- Supplier Consolidation
- Supply Chain and Logistics
- Assembly Line Quality Assurance
- Proactive Maintenance
- Crowdsourced Quality Assurance

Healthcare

- Genomic data for medical trials
- Monitor patient vitals
- Reduce re-admittance rates
- Store medical research data
- Recruit cohorts for pharmaceutical trials

Utilities, Oil & Gas

- Smart meter stream analysis
- Slow oil well decline curves
- Optimize lease bidding
- Compliance reporting
- Proactive equipment repair
- Seismic image processing

Public Sector

- Analyze public sentiment
- Protect critical networks
- Prevent fraud and waste
- Crowdsource reporting for repairs to infrastructure
- Fulfill open records requests

Objectives

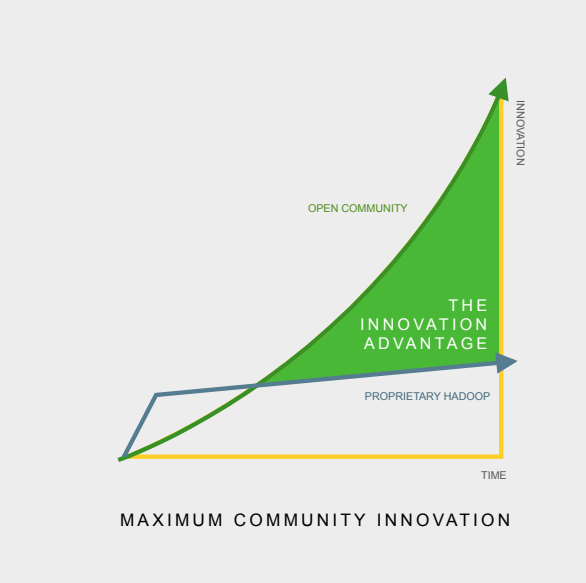
- Data Trends
- Popular Use Cases for Hadoop
- Open Enterprise Hadoop

Open Enterprise Hadoop

- Open
- Central
- Interoperable
- Ready

Open Enterprise Hadoop

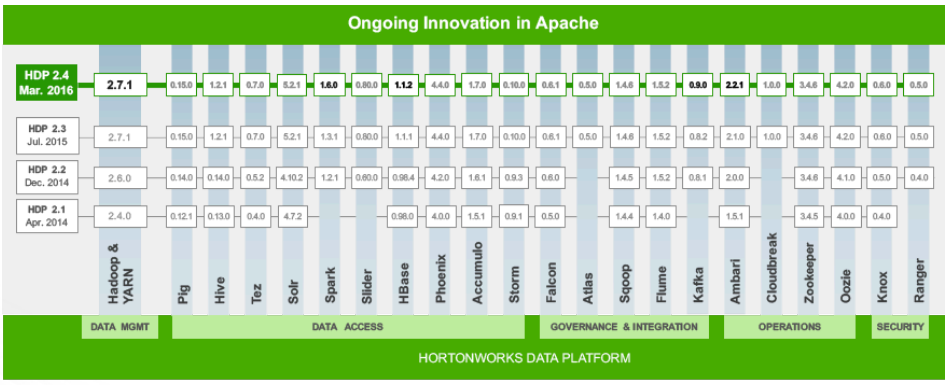
- Open
- Central
- Interoperable
- Ready



Hortonworks Data Platform Is Genuinely Open

- Eliminates Risk
  - of vendor lock-in by delivering 100% Apache open source technology
- Maximizes Community Innovation
  - with hundreds of developers across hundreds of companies
  - **Integrates Seamlessly**
  - through committed co-engineering partnerships with other leading technologies

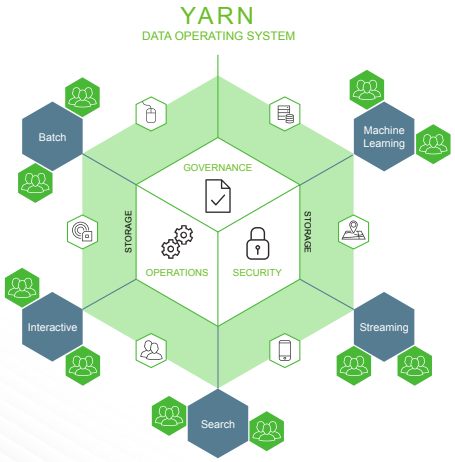
100% Open Approach = Fastest Path to Innovation



Open
Enterprise
Hadoop

- Open
- Central
- Interoperable
- Ready

Centralized Platform with YARN-Based Architecture



Centralized Platform

for operations, governance and security

Diverse Applications

run simultaneously on a single cluster

Maximum Data Ingest

including existing and new sources, regardless of raw format

Shared Big Data Assets

across business groups, functions and users

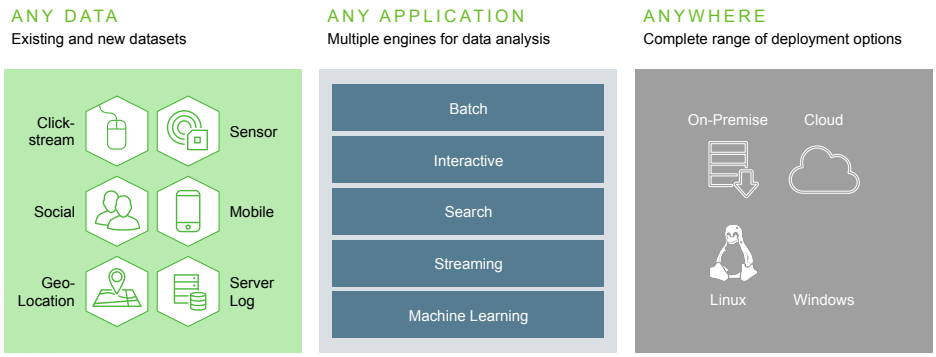
Benefits of the YARN-Based Architecture

	OPEN ENTERPRISE HADOOP 100% YARN-Based Architecture	PROPRIETARY HADOOP Architecture in Silos
Consistent Services (Governance, Security, Operations)	**Confidence with consistent polices for cluster and data management**	Fragmented architecture for the key services **increases complexity and risk**
Resource Efficiency (Management, Hardware, People)	**Shared storage and processing minimizes total cost of ownership**	Redundant clusters mean **more hardware, more data movement,** and **more cost**
Ease of Expansion (Engines, Applications, Clusters)	**Streamlined cluster deployment and also a steady stream of new big data apps to run on YARN**	New applications require additional clusters which **slows deployment and integration**

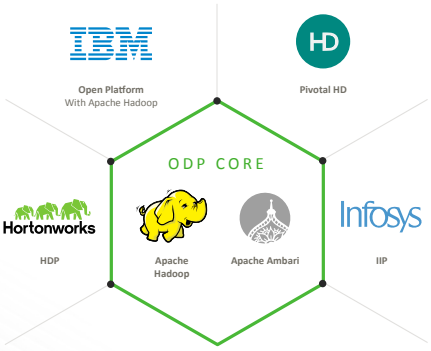
Open Enterprise Hadoop

- Open
- Central
- Interoperable
- Ready

Offering You the Most Flexibility



Synchronized with Industry Standards



Improves Ecosystem Interoperability

as part of the Open Data Platform (ODP) initiative, founded by Hortonworks

Unlocks Choice

for the customer to use components from multiple vendors integrated with HDP

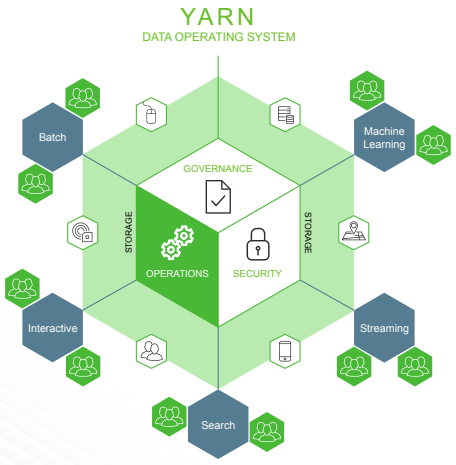
Eliminates Wasteful Guesswork

for the architect who needs to coordinate system versions

Open
Enterprise
Hadoop

- Open
- Central
- Interoperable
- Ready

Provides Consistent Operations



Centralized

management and monitoring of Hadoop clusters

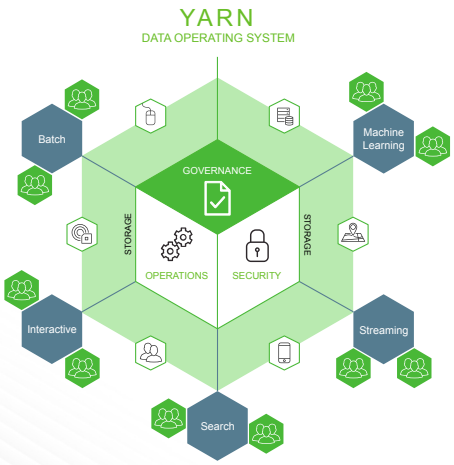
Automated Provisioning

either on-premises or in the cloud with the Cloudbreak API for clusters in minutes

Managed Services

for high availability and consistent lifecycle controls, with dashboards and alerts

Enables Trusted Governance



Data Management

along the entire data lifecycle

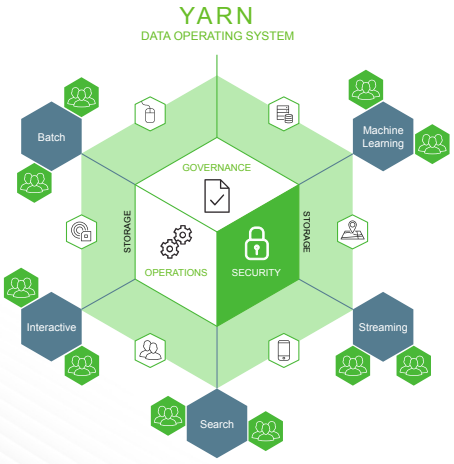
Modeling with Metadata

enables comprehensive data lineage through a hybrid approach

Interoperable Solutions

across the Hadoop ecosystem, through a common metadata store

Ensures Comprehensive Security



Comprehensive Security

through a platform approach

Encrypted Data

at rest and in motion

Centralized Administration

of security policies and user authentication

Fine-Grain Authorization

for data access control

Agile Analytics with Enterprise Spark at Scale

Powering Agile Analytics

via data science notebooks and automation for most common analytics (including geospatial and entity resolution)

Seamless Data Access

across as many data types as possible

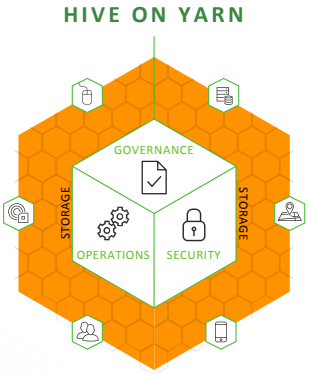
Unmatched Economics

Combining in-memory processing speed with HDP's cost efficiencies at scale

Ready for the Enterprise

with robust security, governance and operations coordinated centrally by Apache Hadoop and YARN

Fast SQL with Apache Hive



Pluggable Architecture

supports Apache Hive, Pivotal HAWQ and other leading SQL engines

Familiar SQL Query Semantics

enable transactions and SQL:2011 Analytics for rich reporting

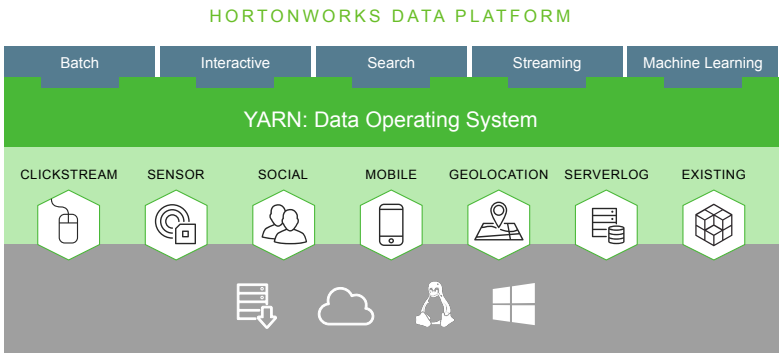
Unprecedented Speed at Extreme Scale

returns query results in interactive time, even as data sets grow to petabytes

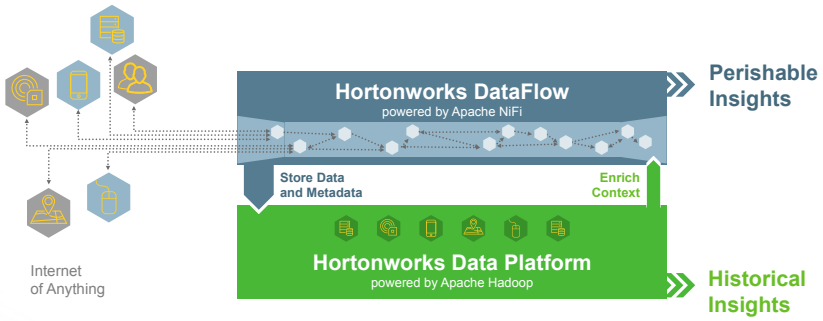
Objectives

- Data Trends
- Popular Use Cases for Hadoop
- Open Enterprise Hadoop
- Why Hortonworks?

Only Hortonworks Delivers Open Enterprise Hadoop



Hortonworks DataFlow Adds to Hadoop Capabilities



Hortonworks DataFlow and Hortonworks Data Platform deliver the industry's most complete solution for Big Data management

Knowledge Check

Questions

1. List and explain what the “3 V’s” are.
2. What are the next two data size classifications beyond Petabyte?
3. What organization manages Hadoop and its ecosystem of tools and frameworks?
4. Identify two of the six common use case families.
5. What one of these use case families is the most widely sought after?

Summary

Summary

- The 3V's of Big Data are driving the adoption of Apache Hadoop (44 ZB by 2020)
- Existing data architectures make data inaccessible, incomplete, irrelevant, and expensive
- Hadoop is a scalable, fault tolerant, open source framework for the distributed storing and processing of large sets of data on commodity hardware
- Six common use case families have emerged
  - Data Discovery
  - Single View
  - Predictive Analytics
  - Active Archive
  - ETL Offload
  - Data Enrichment
- YARN-centralized HDP = Open Enterprise Hadoop

Lesson Objectives

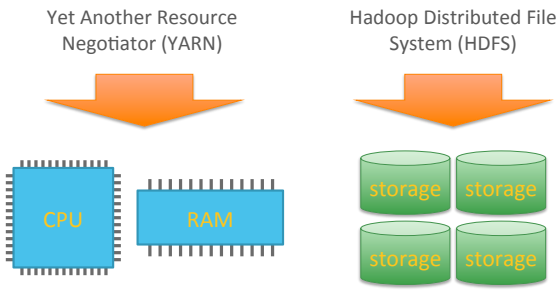
After completing this lesson, students should be able to:

- Describe the Hadoop ecosystem frameworks across the following five architectural categories:
  - Data Management
  - Data Access
  - Data Governance & Integration
  - Security
  - Operations
- Deploy Hadoop into a datacenter – Connected Data Platforms
  - Hadoop cluster node types
  - Integrating with existing data applications
- Observe the demonstration: *Operational Overview with Ambari*

Objectives

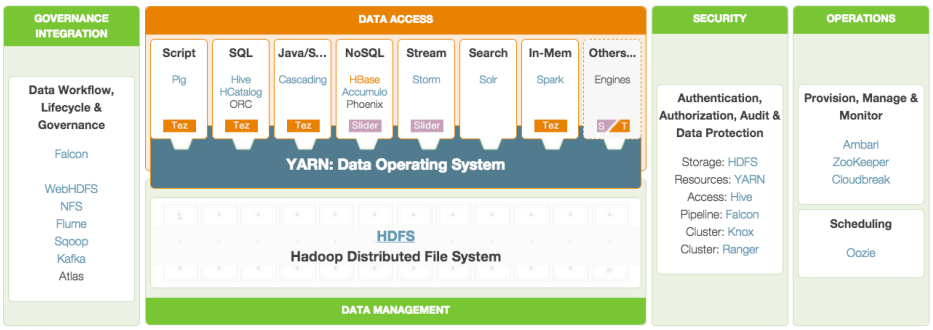
- Hadoop Ecosystem Frameworks

Hadoop Core = Storage + Compute



The Hadoop Ecosystem

Hortonworks Hadoop Distribution



Data Management Frameworks

Framework	Description
Hadoop Distributed File System (HDFS)	A Java-based, distributed file system that provides scalable, reliable, high-throughput access to application data stored across commodity servers
Yet Another Resource Negotiator (YARN)	A framework for cluster resource management and job scheduling

Operations Frameworks

Framework	Description
Ambari	A Web-based framework for provisioning, managing, and monitoring Hadoop clusters
ZooKeeper	A high-performance coordination service for distributed applications
Cloudbreak	A tool for provisioning and managing Hadoop clusters in the cloud
Oozie	A server-based workflow engine used to execute Hadoop jobs

Data Access Frameworks

Framework	Description
Pig	A high-level platform for extracting, transforming, or analyzing large datasets
Hive	A data warehouse infrastructure that supports ad hoc SQL queries
HCatalog	A table information, schema, and metadata management layer supporting Hive, Pig, MapReduce, and Tez processing
Cascading	An application development framework for building data applications, abstracting the details of complex MapReduce programming
HBase	A scalable, distributed NoSQL database that supports structured data storage for large tables
Phoenix	A client-side SQL layer over HBase that provides low-latency access to HBase data
Accumulo	A low-latency, large table data storage and retrieval system with cell-level security
Storm	A distributed computation system for processing continuous streams of real-time data
Solr	A distributed search platform capable of indexing petabytes of data
Spark	A fast, general purpose processing engine use to build and run sophisticated SQL, streaming, machine learning, or graphics applications

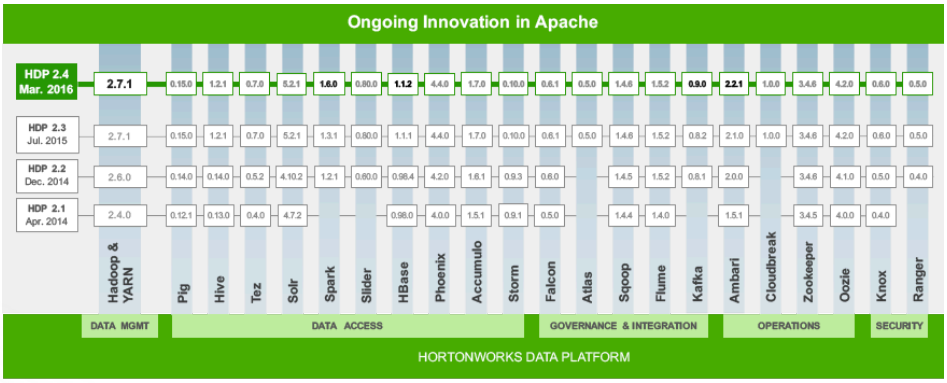
Governance and Integration Frameworks

Framework	Description
Falcon	A data governance tool providing workflow orchestration, data lifecycle management, and data replication services.
WebHDFS	A REST API that uses the standard HTTP verbs to access, operate, and manage HDFS
HDFS NFS Gateway	A gateway that enables access to HDFS as an NFS mounted file system
Flume	A distributed, reliable, and highly-available service that efficiently collects, aggregates, and moves streaming data
Sqoop	A set of tools for importing and exporting data between Hadoop and RDBM systems
Kafka	A fast, scalable, durable, and fault-tolerant publish-subscribe messaging system
Atlas	A scalable and extensible set of core governance services enabling enterprises to meet compliance and data integration requirements

Security Frameworks

Framework	Description
HDFS	A storage management service providing file and directory permissions, even more granular file and directory access control lists, and transparent data encryption
YARN	A resource management service with access control lists controlling access to compute resources and YARN administrative functions
Hive	A data warehouse infrastructure service providing granular access controls to table columns and rows
Falcon	A data governance tool providing access control lists that limit who may submit Hadoop jobs
Knox	A gateway providing perimeter security to a Hadoop cluster
Ranger	A centralized security framework offering fine-grained policy controls for HDFS, Hive, HBase, Knox, Storm, Kafka, and Solr

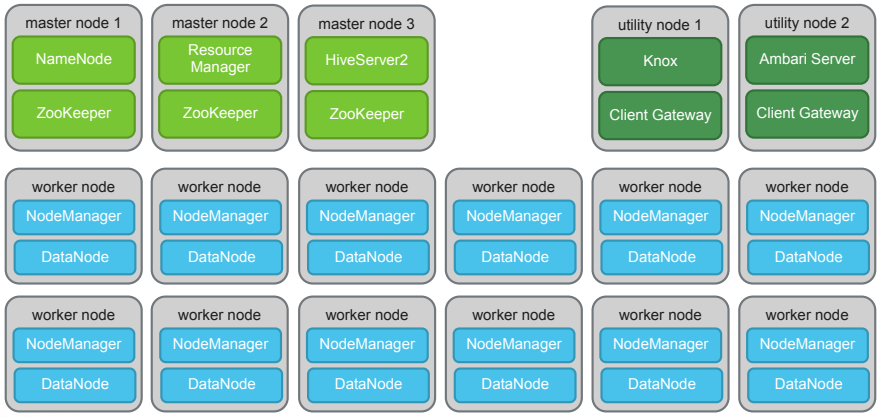
Ecosystem Component Versions



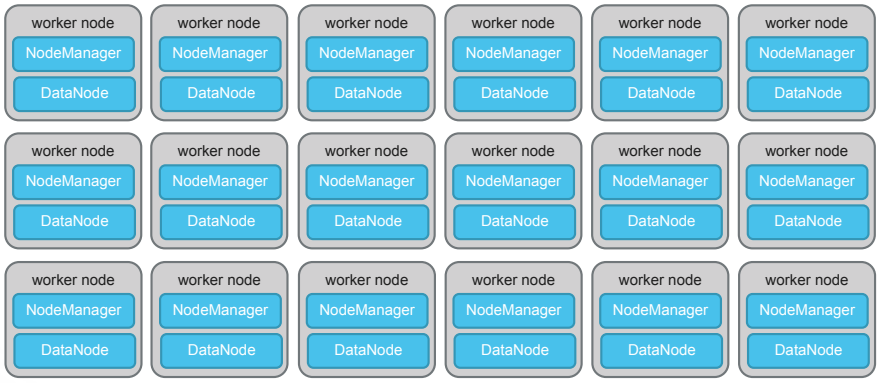
Objectives

- Hadoop Ecosystem Frameworks
- Hadoop in the Datacenter

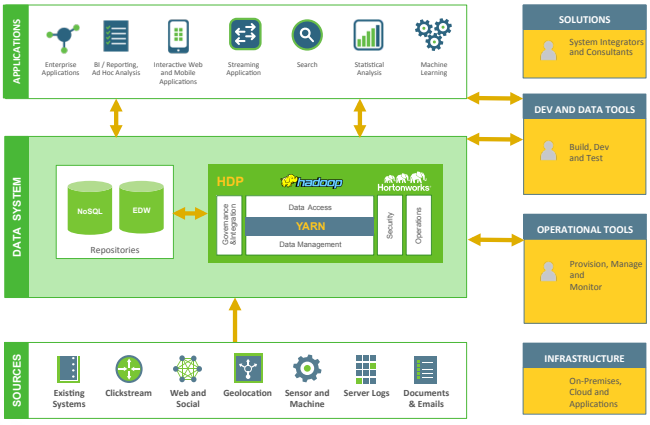
Distinct Masters and Scale-Out Workers



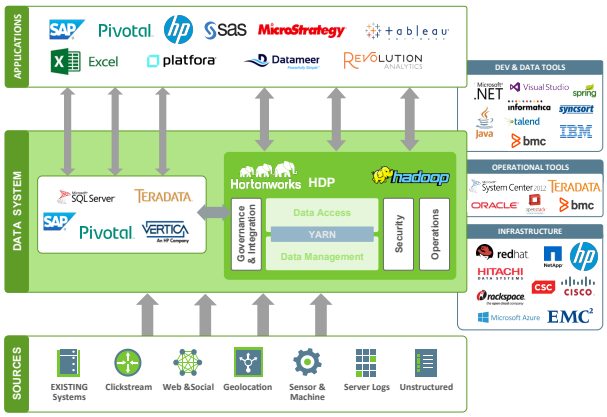
Worker Nodes can Scale into the Thousands



Connected Data Platforms



Hadoop as a +1 Architecture



Knowledge Check

Questions

1. Match the following components with the architectural categories of Data Management, Data Access, Governance & Integration, Security, and Operations
   - Ambari
   - HBase
   - HDFS
   - Sqoop
   - Ranger
2. True/False? The number of master nodes grows in proportion to the number of workers.
3. List a few types of data sources that are new to most organizations.
4. True/False? Hadoop's goal is to displace all existing data systems.

Summary

Summary

- Hadoop ecosystem frameworks fall into the following five categories:
  - Data Management
  - Data Access
  - Governance & Integration
  - Security
  - Operations
- Primary server stereotypes are:
  - Master nodes
  - Worker nodes
- Hadoop complements existing systems and is the foundation of Connected Data Platforms

Demo: Operational Overview with Ambari

Lesson Objectives

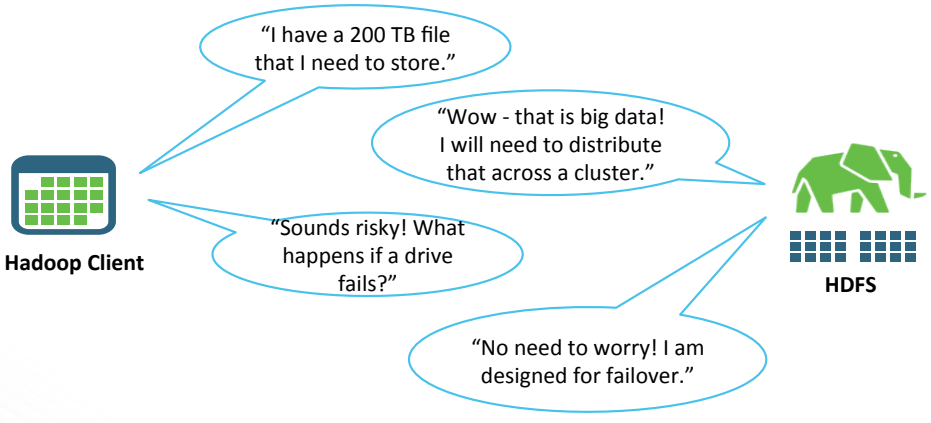
After completing this lesson, students should be able to:

- Present an overview of the Hadoop Distributed File System (HDFS)
- Detail the major architectural components and their interactions
  - NameNode
  - DataNode
  - Clients
- Discuss additional features
- Observe the demonstration: *Loading Data into HDFS*

Objectives

- HDFS Overview

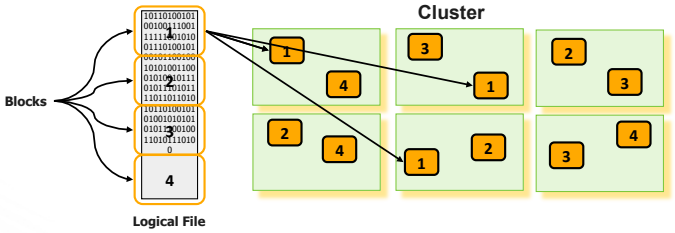
What is HDFS?



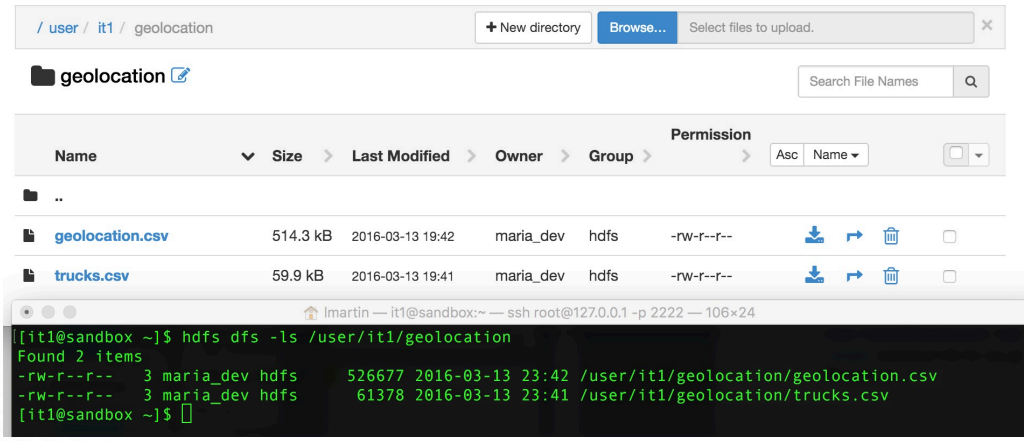
HDFS

Key Ideas

- Write Once, Read Many times (WORM)
- Divide files into big blocks and distribute across the cluster
- Store multiple replicas of each block for reliability
- Programs can ask "where do the pieces of my file live?"



It Looks Like a File System



It Acts Like a File System

```
hdfs dfs –command [args]
```

- A few of the almost 30 HDFS commands:

  `-cat`: display file content (uncompressed)
  `-text`: just like cat but works on compressed files
  `-chgrp`,`-chmod`,`-chown`: changes file permissions
  `-put`,`-get`,`-copyFromLocal`,`-copyToLocal`: copies files from the local file system to the HDFS and vice versa.
  `-ls`, `-ls -R`: list files/directories
  `-mv`,`-moveFromLocal`,`-moveToLocal`: moves files
  `-stat`: statistical info for any given file (block size, number of blocks, file type, etc.)

Objectives

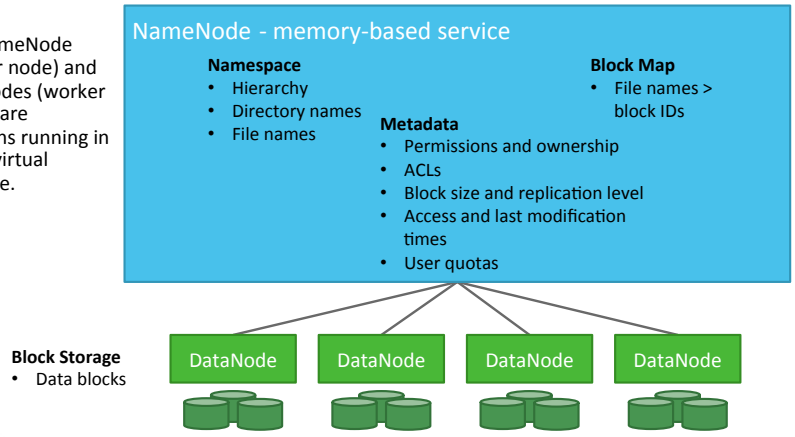
- HDFS Overview
- HDFS Components and Interactions

HDFS Components

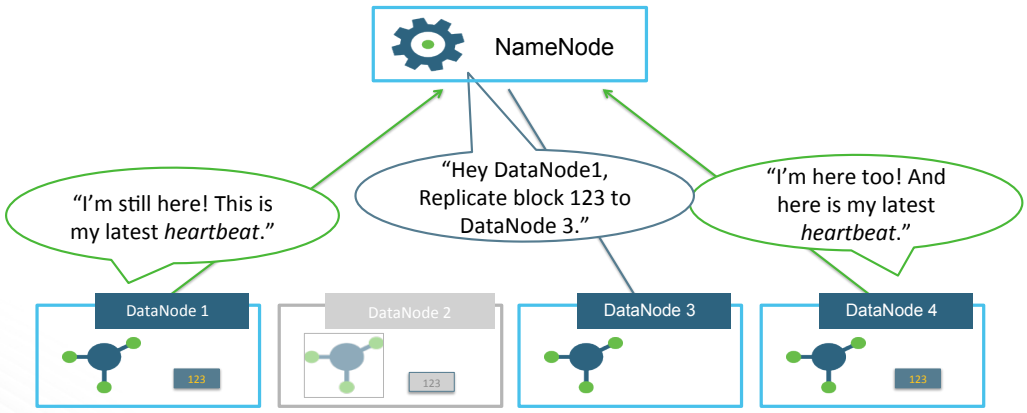
- **NameNode**
  - Is the master service of HDFS
  - Determines and maintains how the chunks of data are distributed across the DataNodes
- **DataNode**
  - Stores the chunks of data, and is responsible for replicating the chunks across other DataNodes

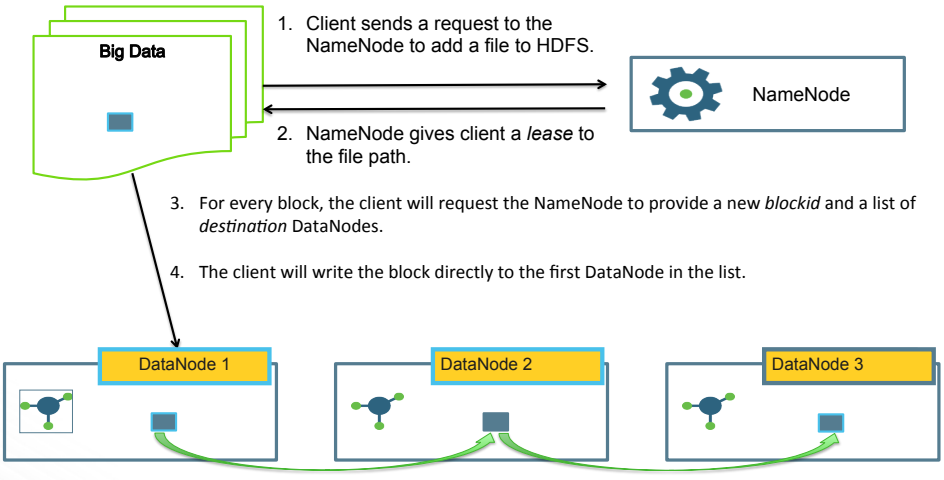
HDFS Architecture

- The NameNode (master node) and DataNodes (worker nodes) are daemons running in a Java virtual machine.

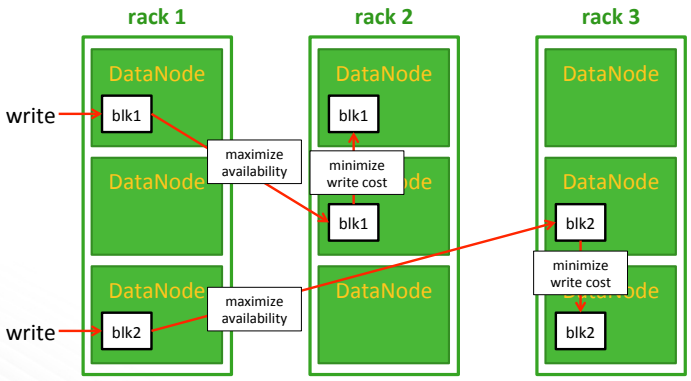


The DataNodes

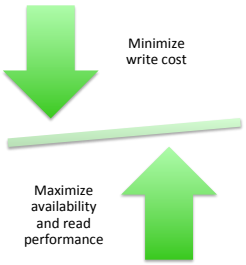




Replication and Block Placement



HDFS is designed to assume, and handle, disk and system failures.



Objectives

- HDFS Overview
- HDFS Components and Interactions
- Additional HDFS Interactions

NameNode High Availability

- The HDFS NameNode is a single point of failure.
  - The entire cluster is unavailable if the NameNode:
    - Fails or becomes unreachable
    - Is stopped to perform maintenance
- NameNode HA:
  - Uses a redundant NameNode
  - Is configured in an Active/Standby configuration
  - Enables fast failover in response to NameNode failure
  - Permits administrator-initiated failover for maintenance
  - Is configured by Ambari

Heterogeneous Storage

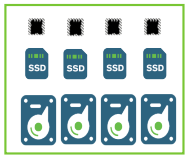
Before

- DataNode is a single storage
- Storage is uniform - Only storage type Disk
- Storage types hidden from the file system

All disks as a single storage

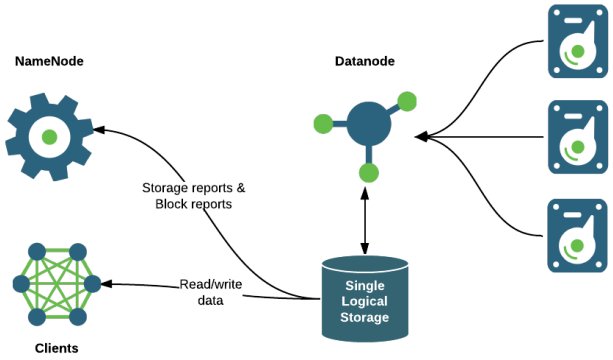
New Architecture

- DataNode is a collection of storages
- Support different types of storages
- Disk, SSDs, Memory

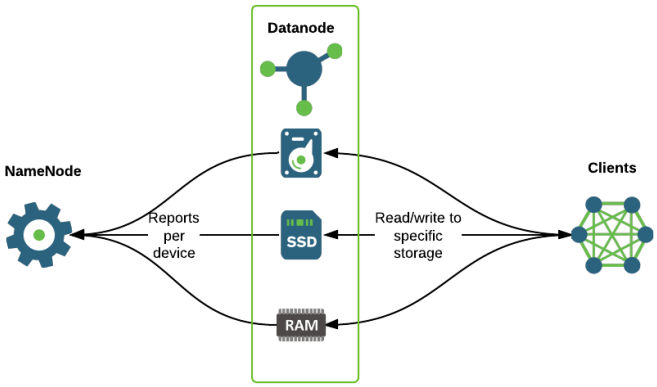


Collection of tiered storages

HDFS Storage Before



HDFS Storage Now



HDFS Multi-Tenant Controls

- **Security**
  - Classic POSIX permissioning (ex: -rwxr-xr--)
  - Extended Access Control Lists (ACL) for richer scenarios
  - Centralized authorization policies and audit available via Ranger plug-in
- **Quotas**
  - Easy to understand data size quotas
  - Additional option for controlling the number of files

Knowledge Check

Questions

1. HDFS breaks files into _____ and persists multiple _____ across the cluster to aid in the file system's _____ and the to help programs obtain _____.
2. What is the primary master node service?
3. What is the worker node service?
4. True/False? Clients avoid writing data through the NameNode.
5. True/False? Clients write replica copies directly to each DataNode.

Summary

Summary

- HDFS breaks files into blocks and replicates them for reliability and processing data locality
- The primary components are the master NameNode service and the worker DataNode service
- The NameNode is a memory-based service
- The NameNode automatically takes care of recovery missing and corrupted blocks
- Clients interact with the NameNode to get a list, for each block, of DataNodes to write data to

Demo: Loading Data into HDFS

Lesson Objectives

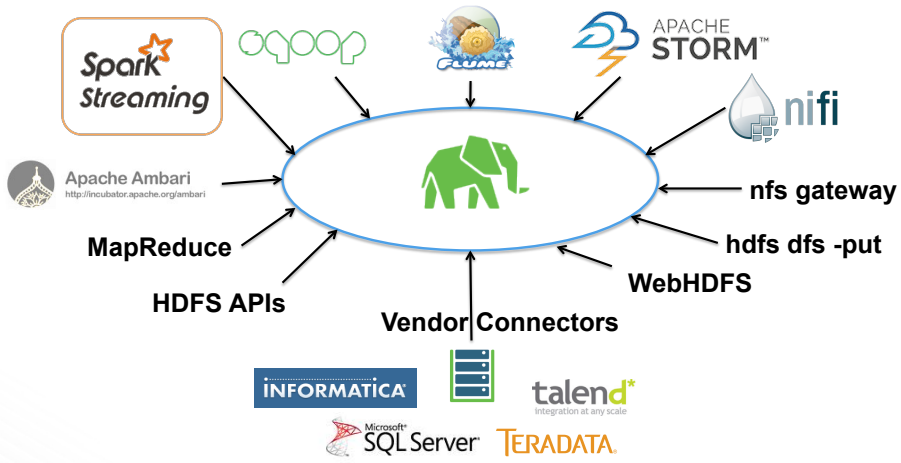
After completing this lesson, students should be able to:

- Describe data ingestion
- Describe Batch/Bulk ingestion options
  - Ambari HDFS Files View
  - CLI & WebHDFS
  - NFS Gateway
  - Sqoop
- Describe streaming framework alternatives
  - Flume
  - Storm
  - Spark Streaming
  - HDF / NiFi
- Observe the demonstration: *Streaming Data into HDFS* (Time Permitting)

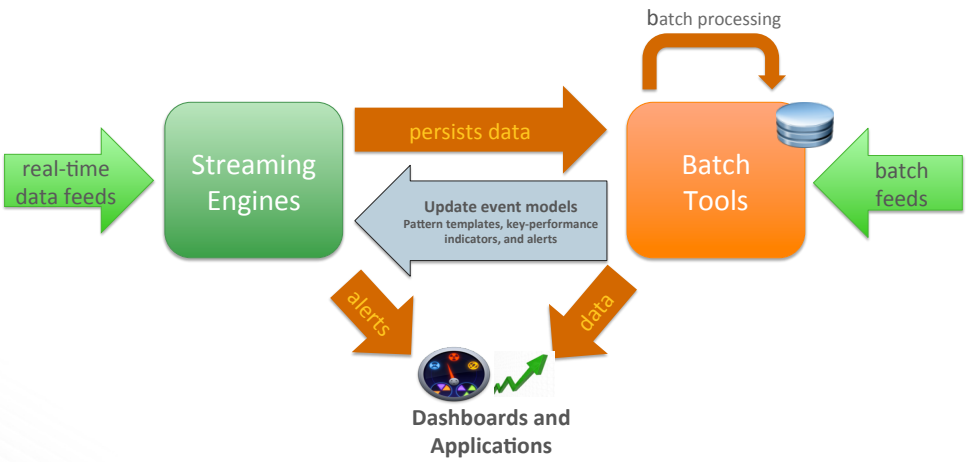
- Ingestion Overview

Objectives

Data Input Options



Multiple Ingestion Workflows – Batch and Real-Time



Real-Time Versus Batch Ingestion Workflows

Real-time and batch processing are very different.

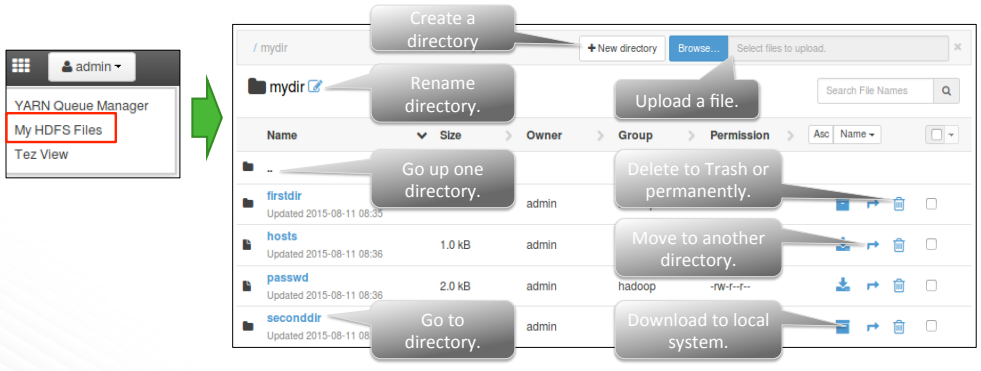
Factors		Real-Time	Batch
Data	Age	Real-time – usually less than 15 minutes old	Historical – usually more than 15 minutes old
	Location	Primarily in memory – moved to disk after processing	Primarily on disk – moved to memory for processing
Processing	Speed	Sub-second to few seconds	Few seconds to hours
	Frequency	Always running	Sporadic to periodic
Clients	Who	Automated systems only	Human & automated systems
	Type	Primarily operational applications	Primarily analytical applications

Objectives

- Ingestion Overview
- Batch/Bulk Ingestion

Ambari Files View

The Files View is an Ambari Web UI plug-in providing a graphical interface to HDFS.



The Hadoop Client

- The `put` command to uploading data to HDFS
- Perfect for inputting local files into HDFS
- Useful in batch scripts
- Usage:

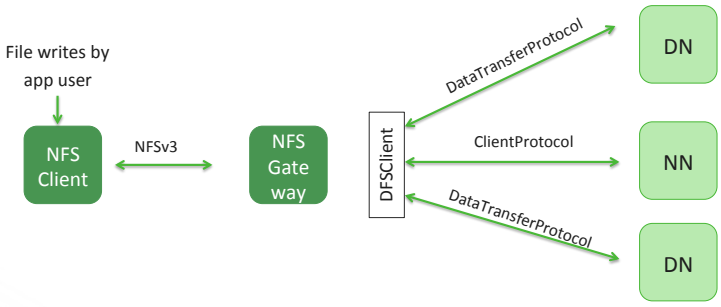
```
hdfs dfs –put mylocalfile /some/hdfs/path
```

WebHDFS

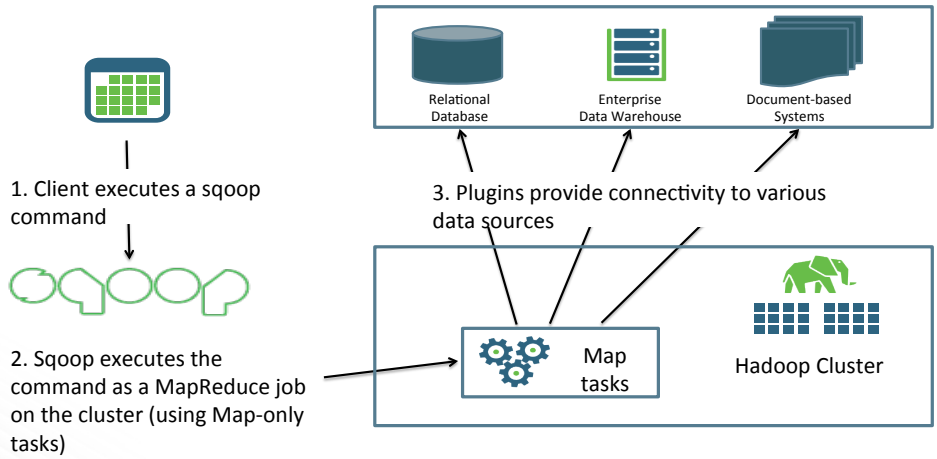
- REST API for accessing all of the HDFS file system interfaces:
  - `http://host:port/webhdfs/v1/test/mydata.txt?op=OPEN`
  - `http://host:port/webhdfs/v1/user/train/data?op=MKDIRS`
  - `http://host:port/webhdfs/v1/test/mydata.txt?op=APPEND`

NFS Gateway

- Uses NFS standard and supports all HDFS commands
- No random writes



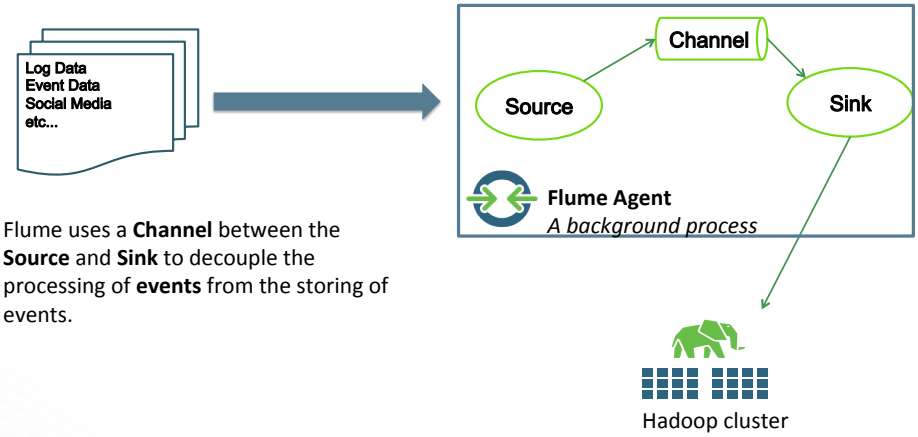
Sqoop: Database Import/Export



Objectives

- Ingestion Overview
- Batch/Bulk Ingestion
- Streaming Alternatives

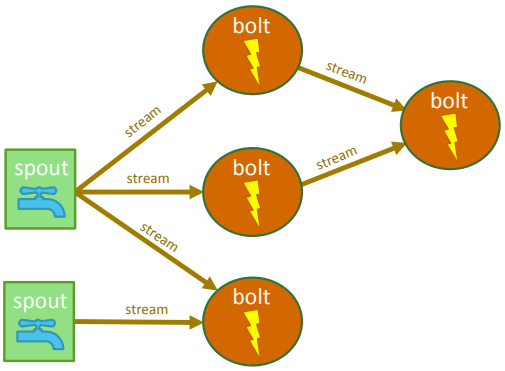
Flume: Data Streaming



Flume uses a **Channel** between the **Source** and **Sink** to decouple the processing of **events** from the storing of events.

Storm Topology Overview

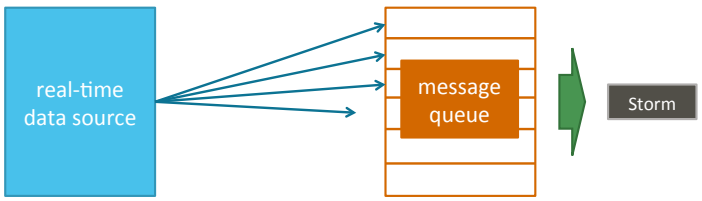
- Storm data processing occurs in a topology.
- A topology consists of spout and bolt components.
- Spouts bring data into the topology
- Bolts can (not required) persist data including to HDFS



Storm topology

Message Queues

Various types of message queues are often the source of the data processed by real-time processing engines like Storm



operating systems, services and applications, sensors

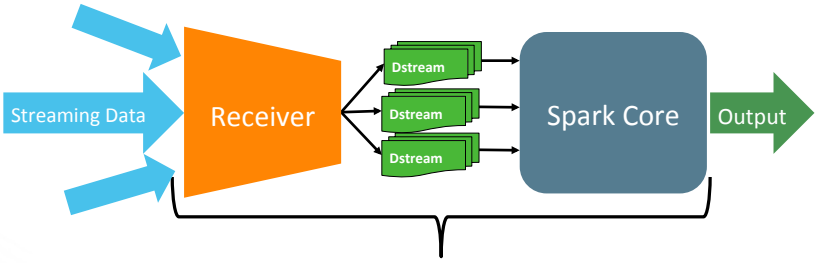
log entries, events, errors, status messages, etc.

Kestrel, RabbitMQ, AMQP, Kafka, JMS, others…

data from queue is read by Storm

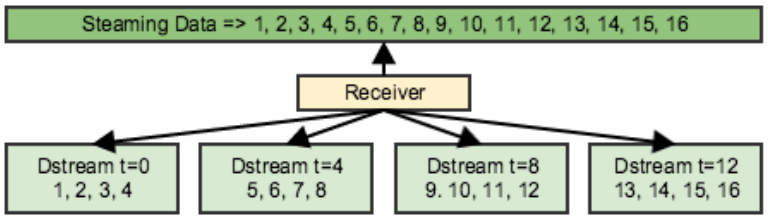
Spark Streaming

- Streaming Applications consist of the same components as a Core application, but add the concept of a receiver
- The receiver is a process running on an executor

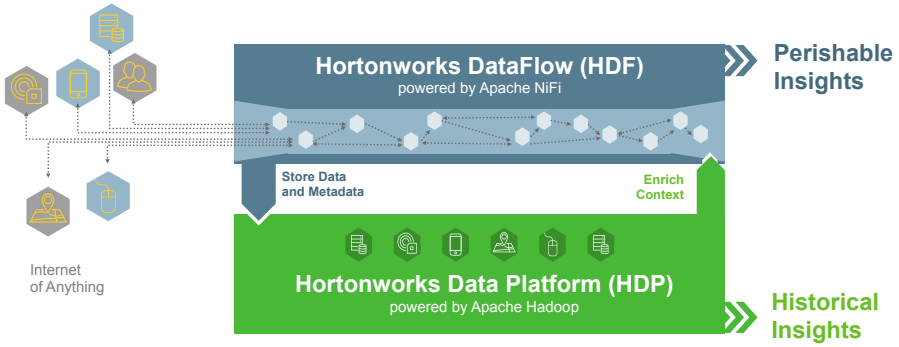


Spark Streaming's Micro-Batch Approach

- Micro-batches are created at regular time intervals
  - Receiver takes the data and starts filling up a batch
  - After the batch duration completes, data is shipped off
  - Each batch forms a collection of data entities that are processed together



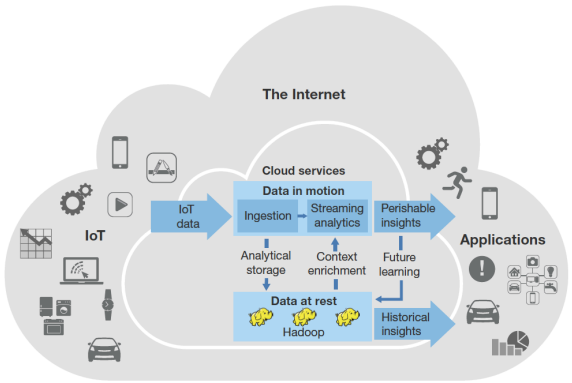
HDF with HDP – A Complete Big Data Solution



Hortonworks DataFlow and the Hortonworks Data Platform deliver the industry's most complete Big Data solution

HDF Feeds Data to Streaming Analytics in HDP

Figure 1 IoT App Design Must Simultaneously Support Analytics In Motion And Analytics At Rest



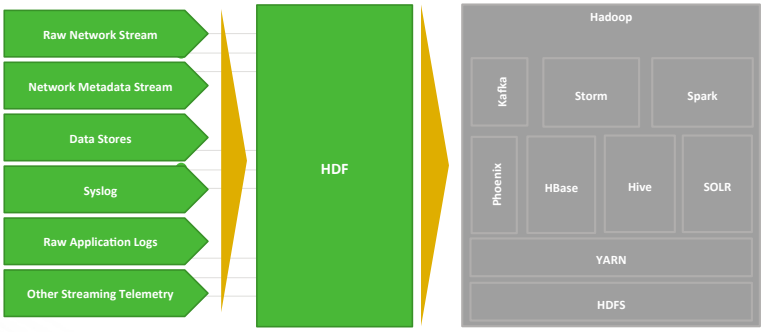
Hortonworks DataFlow for perishable insights from data-in-motion

Hortonworks Data Platform for rich historical insights from data-at-rest

Source: Forrester - Internet Of Things Applications Hunger For Hadoop And Real-Time Analytics In The Cloud, March 2015

Big Data Ingestion with HDF

HDF workflows and Storm/Spark streaming workflows can be coupled



Knowledge Check

Questions

1. What tool is used for importing data from a RDBMS?
2. List two ways to easily script moving files into HDFS.
3. Identify a major client limitation imposed by the NFS Gateway.
4. True/False? Storm operates on micro-batches.
5. Name the popular open-source messaging component that is bundled with HDP.

Summary

Summary

- There are many different ways to ingest data including customer solutions written via HDFS APIs as well as vendor connectors
- Streaming and batch workflows can work together in a holistic system
- The NFS Gateway may help some legacy systems populate data into HDFS
- Sqoop's configurable number of database connection can overload an RDBMS
- The following are streaming frameworks:
  - Flume
  - Storm
  - Spark Streaming
  - HDF / NiFi

Demo: Streaming Data into HDFS

Lesson Objectives

After completing this lesson, students should be able to:

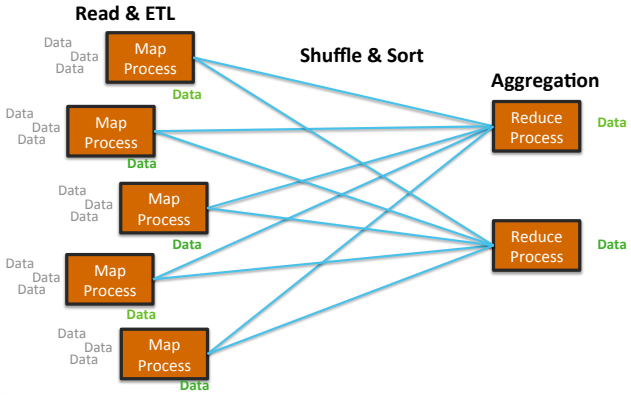
- Describe how MapReduce works
  - Explain the reliance on the Key Value Pair (KVP) paradigm
  - Illustrate the MapReduce framework with simple examples
- Observe the demonstration: *Processing with MapReduce* (Time Permitting)

Objectives

- MapReduce Framework

What is MapReduce?

Breaking a large problem into sub-solutions



Simple Algorithm

1. Review stack of quarters
2. Count each year that ends in an even number

Processing at Scale

Distributed Algorithm – MapReduce

Map
(total number of quarters)

Reduce
(sum each person's total)

The Mapper

- **The Mapper reads data in the form of key/value pairs (KVPs)**
- **It outputs zero or more KVPs**
- **The Mapper may use or completely ignore the input key**
  - For example, a standard pattern is to read a line of a file at a time
    - The key is the byte offset into the file at which the line starts
    - The value is the contents of the line itself
    - Typically the key is considered irrelevant with this pattern
- **If the Mapper writes anything out, it must in the form of KVPs**
  - This "intermediate data" is NOT stored in HDFS (local storage only without replication)

The Reducer

- **After the Map phase is over, all the intermediate values for a given intermediate key are combined together into a list**
- **This list is given to a Reducer**
  - There may be a single Reducer, or multiple Reducers
  - All values associated with a particular intermediate key are guaranteed to go to the same Reducer
  - The intermediate keys, and their value lists, are passed in sorted order
- **The Reducer outputs zero or more KVPs**
  - These are written to HDFS
  - In practice, the Reducer often emits a single KVP for each input key

MapReduce Example – Word Count

Count the number of occurrences of each word in a large amount of input data

```
map(String input_key, String input_value)
      foreach word in input_value:
            emit(w,1)
```

```
reduce(String output_key, Iter<int> intermediate_vals)
      set count = 0
      foreach val in intermediate_vals:
            count += val
      emit(output_key, count)
```

MapReduce Example – Map Phase

Input to Mapper

```
(8675, 'I will not eat green
eggs and ham')

(8709, 'I will not eat them Sam
I am')
```

- Ignoring the key
  - It is just and offset

Output from Mapper

```
('I', 1), ('will', 1), ('not', 1),
('eat', 1), ('green', 1),
('eggs', 1), ('and', 1),
('ham', 1), ('I', 1), ('will', 1),
('not', 1), ('eat', 1),
('them', 1), ('Sam', 1),
('I', 1), ('am', 1)
```

- No attempt is made to optimize within a record in this example
  - This is a great use case for a "Combiner"

MapReduce Example – Reduce Phase

Input to Reducer

```
('I', [1, 1, 1])
('Sam', [1])
('am', [1])
('and', [1])
('eat', [1, 1])
('eggs', [1])
('green', [1])
('ham', [1])
('not', [1, 1])
('them', [1])
('will', [1, 1])
```

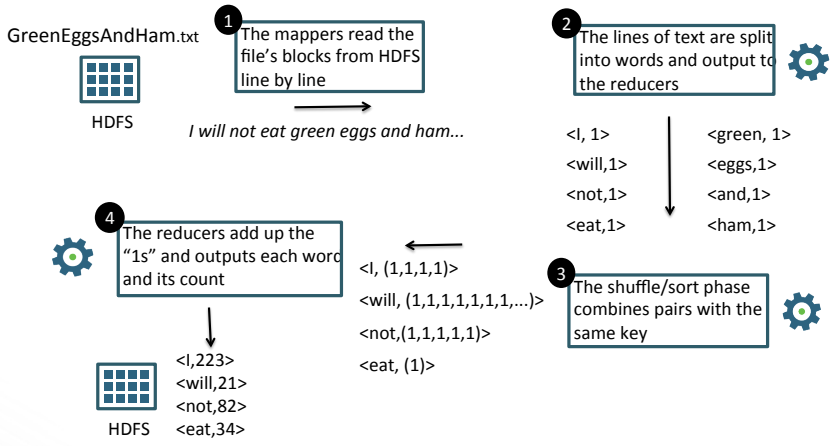
- Notice keys are sorted and associated values for same key are in a single list
  - Shuffle & Sort did this for us

Output from Reducer

```
('I', 3)
('Sam', 1)
('am', 1)
('and', 1)
('eat', 2)
('eggs', 1)
('green', 1)
('ham', 1)
('not', 2)
('them', 1)
('will', 2)
```

- All done!

Recap of Word Count



Knowledge Check

Questions

1. What are the two primary phases of the MapReduce framework? *No, this is not a trick question.*
2. What dictates the number of Mappers that are run? Same question for the Reducers.
3. How many input & output KVPs are passed into, and emitted out of, the Mappers? Same question for the Reducers.
4. True/False? It is possible to have a Reducer-only job.
5. Why were frameworks like Pig and Hive built on top of MapReduce? *Again, not a trick question…*

Summary

Summary

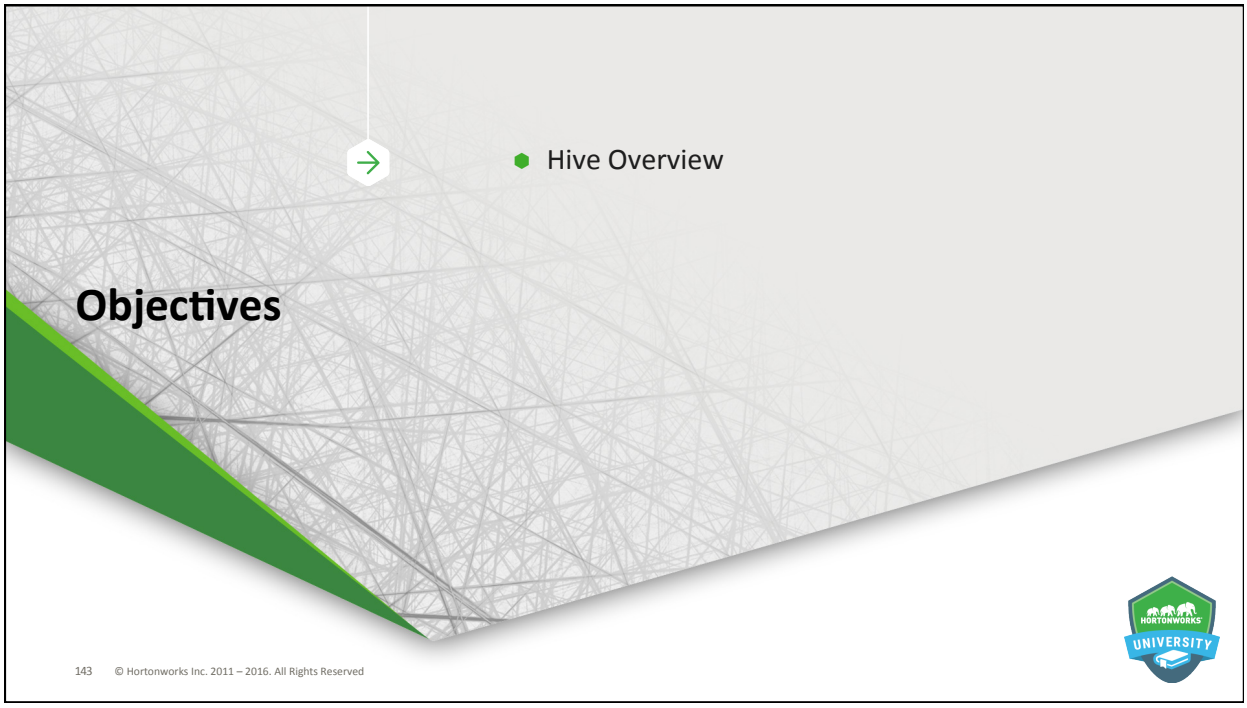
- MapReduce is the foundational framework for processing data at scale because of its ability to break a large problem into any smaller ones
- Mappers read data in the form of KVPs and each call to a Mapper is for a single KVP; it can return 0..m KVPs
- The framework shuffles & sorts the Mappers' outputted KVPs with the guarantee that only one Reducer will be asked to process a given Key's data
- Reducers are given a list of Values for a specific Key; they can return 0..m KVPs
- Due to the fine-grained nature of the framework, many use cases are better suited for higher-order tools

Demo: Processing with MapReduce

Lesson Objectives

After completing this lesson, students should be able to:

- Present an overview of Hive
  - Compare/contrast to RDBMS technologies
  - Step through the architectural design
- Explain how to perform classic operations
  - Crete and populate tables
  - Utilize views
- Review the performance improvements from the Stinger initiatives
- Observe the demonstration: *Data Manipulation with Hive*



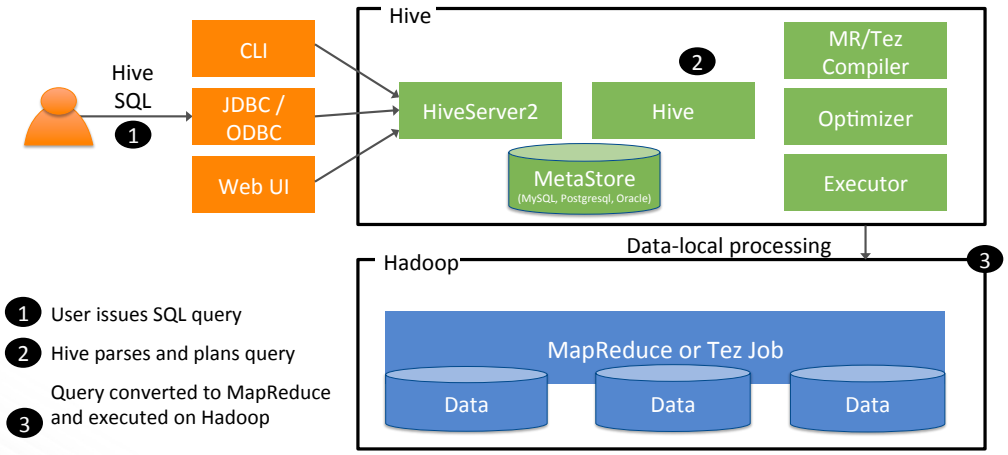
What is Hive?

- Data warehouse system for Hadoop
- Create schemas/table definitions that point to data in Hadoop
- Treat your data in Hadoop as tables
- SQL 92
- Interactive queries at scale

Hive's Alignment with SQL

SQL Datatypes	SQL Semantics
INT	SELECT, LOAD, INSERT from query
TINYINT/SMALLINT/BIGINT	Expressions in WHERE and HAVING
BOOLEAN	GROUP BY, ORDER BY, SORT BY
FLOAT	CLUSTER BY, DISTRIBUTE BY
DOUBLE	Sub-queries in FROM clause
STRING	GROUP BY, ORDER BY
BINARY	ROLLUP and CUBE
TIMESTAMP	UNION
ARRAY, MAP, STRUCT, UNION	LEFT, RIGHT and FULL INNER/OUTER JOIN
DECIMAL	CROSS JOIN, LEFT SEMI JOIN
CHAR	Windowing functions (OVER, RANK, etc.)
VARCHAR	Sub-queries for IN/NOT IN, HAVING
DATE	EXISTS / NOT EXISTS
	INTERSECT, EXCEPT

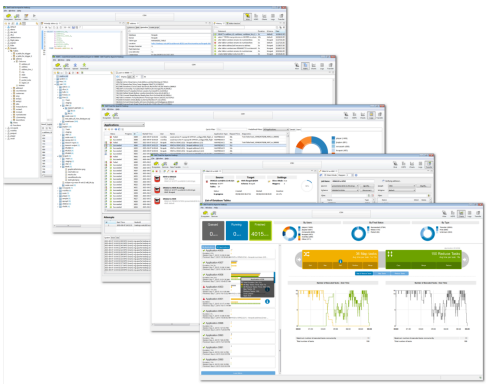
Hive Architecture



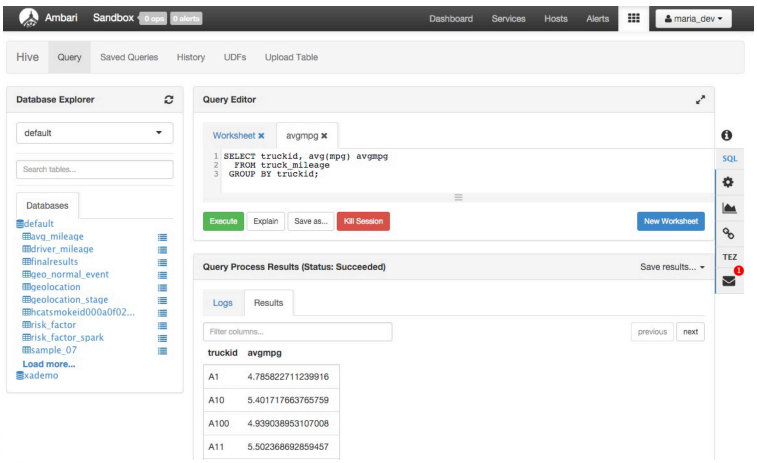
1. User issues SQL query
2. Hive parses and plans query
3. Query converted to MapReduce and executed on Hadoop

Submitting Hive Queries – *CLI and GUI Tools*

```
[it1@sandbox ~]$ beeline -u jdbc:hive2://localhost:10000
WARNING: Use "yarn jar" to launch YARN applications.
Connecting to jdbc:hive2://localhost:10000
Connected to: Apache Hive (version 1.2.1000.2.4.0.0-169)
Driver: Hive JDBC (version 1.2.1000.2.4.0.0-169)
Transaction isolation: TRANSACTION_REPEATABLE_READ
Beeline version 1.2.1000.2.4.0.0-169 by Apache Hive
0: jdbc:hive2://localhost:10000> show tables;
+---------------------------------+--+
|            tab_name             |
+---------------------------------+--+
| avg_mileage                     |
| driver_mileage                  |
| finalresults                    |
| geo_normal_event                |
| geolocation                     |
| geolocation_stage               |
| hcatsmokeid000a0f02_date250116  |
| risk_factor                     |
| risk_factor_spark               |
| sample_07                       |
| sample_08                       |
| truck_mileage                   |
| trucks                          |
| trucks_stage                    |
+---------------------------------+--+
14 rows selected (0.696 seconds)
0: jdbc:hive2://localhost:10000>
```



Submitting Hive Queries – *Ambari Hive View*



Objectives

- Hive Overview
- Performing Operations in Hive

Defining a Hive-Managed Table

```
CREATE TABLE customer (
            customerID INT,
            firstName STRING,
            lastName STRING,
            birthday TIMESTAMP,
  ) ROW FORMAT DELIMITED
    FIELDS TERMINATED BY ',';
```

Defining an External Table

```
CREATE EXTERNAL TABLE salaries (
   gender string,
   age int,
   salary double,
   zip int
 ) ROW FORMAT DELIMITED
   FIELDS TERMINATED BY ',';
```

Defining a Table LOCATION

```
CREATE EXTERNAL TABLE SALARIES (
   gender string,
   age int,
   salary double,
   zip int
 ) ROW FORMAT DELIMITED
   FIELDS TERMINATED BY ','
   LOCATION '/user/train/salaries/';
```

Loading Data into Hive

```
LOAD DATA LOCAL INPATH ’/tmp/customers.csv' OVERWRITE INTO TABLE
customers;
```

```
LOAD DATA INPATH '/user/train/customers.csv' OVERWRITE INTO TABLE
customers;
```

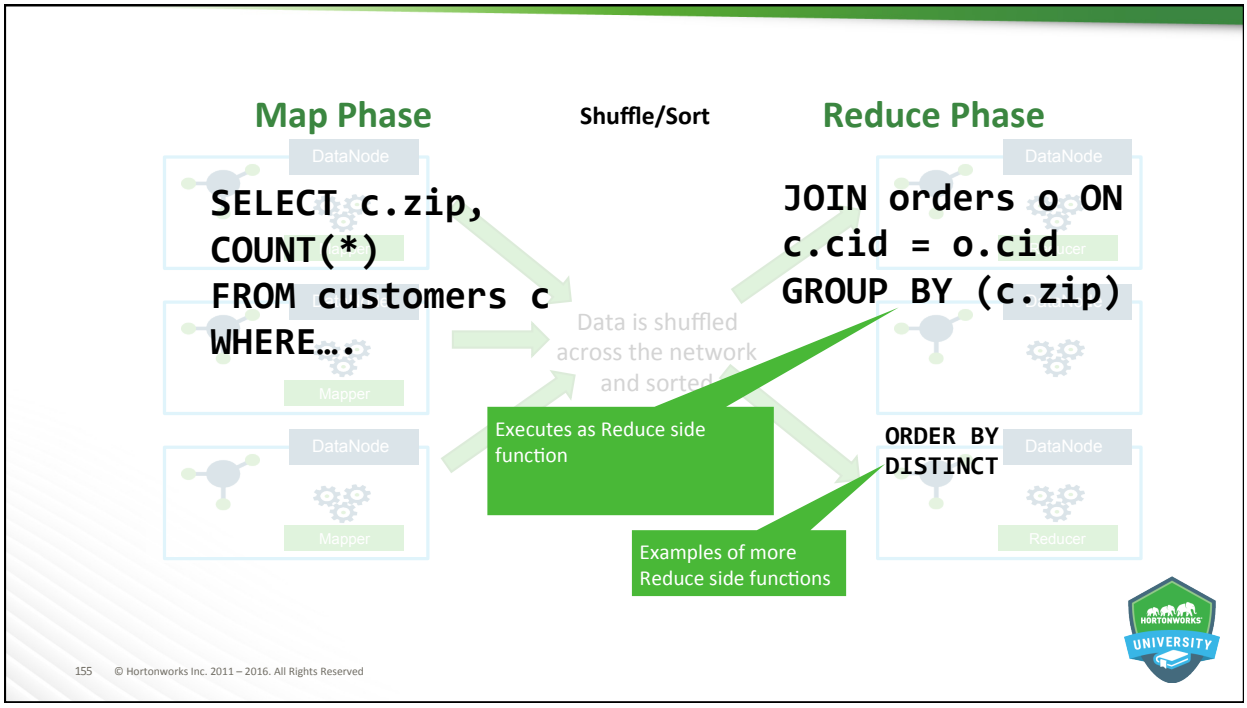
```
INSERT INTO birthdays
    SELECT firstName, lastName, birthday
    FROM customers
    WHERE birthday IS NOT NULL;
```

Performing Queries

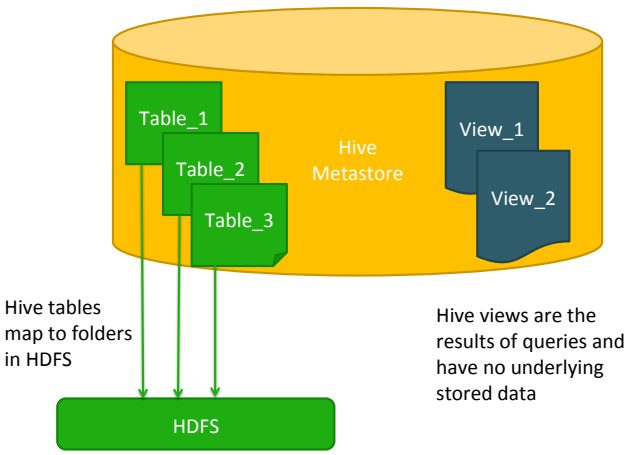
```
SELECT * FROM customers;
```

```
FROM customers
    SELECT firstName, lastName, address, zip
    WHERE orderID > 0
    GROUP BY zip;
```

```
SELECT customers.*, orders.*
    FROM customers
    JOIN orders ON
    (customers.customerID = orders.customerID);
```



Hive Views



Defining Views

```
CREATE VIEW 2010_visitors AS
  SELECT fname, lname,
     time_of_arrival, info_comment
  FROM wh_visits
  WHERE
  cast(substring(time_of_arrival,6,4)
AS int) >= 2010
  AND
  cast(substring(time_of_arrival,6,4)
AS int) < 2011;
```

Using Views

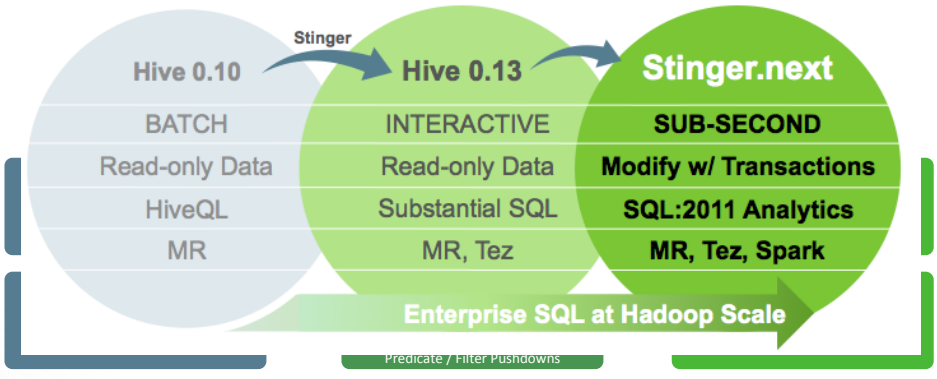
You use a view just like a table:

```
from 2010_visitors
  select *
  where info_comment like "%CONGRESS%"
  order by lname;
```

Objectives

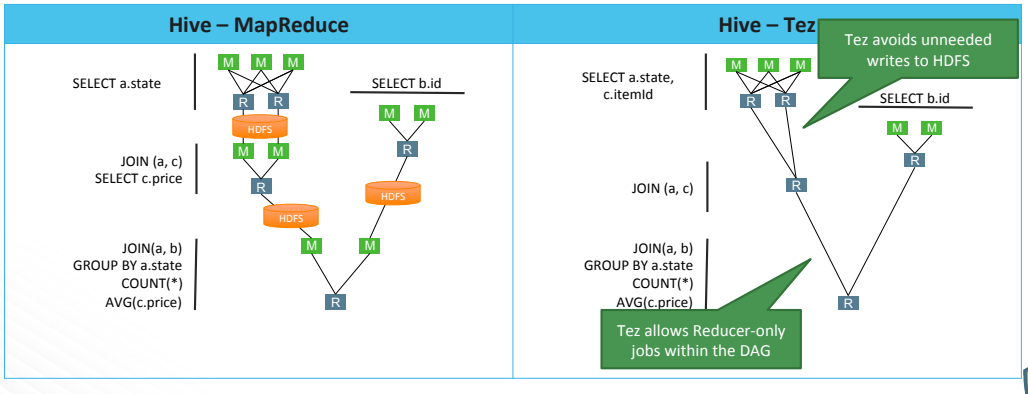
- Hive Overview
- Performing Operations in Hive
- Performance Improvements

Interactive Hive – Overview of Stinger



Tez

```
SELECT a.state, COUNT(*), AVG(c.price)
FROM a
JOIN b ON (a.id = b.id)
JOIN c ON (a.itemId = c.itemId)
GROUP BY a.state
```



Knowledge Check

Questions

1. What element within the Hive architecture do clients make xDBC connections to?
2. List the execution engines that can be used when running your queries.
3. What is the primary difference between an external table and a managed one?
4. True/False? A managed table can be backed by data that resides somewhere other than `/apps/hive/warehouse`.
5. List at least two of the improvements from the original Stinger initiative.

Summary

Summary

- Hive is the data warehouse system for Hadoop and uses the familiar table and SQL metaphors that are used with classic RDBMS solutions
- The MetaStore maintains the logical view of tables as well as the physical characteristics such as where the data is stored and in what format it is in
- Clients, using xDBC, connect to the HiveServer2 component on a master node which in turn submits queries into the worker nodes for processing
- Hive can create, populate and query tables
- Views are supported, but they are not materialized
- Significant performance improvements have surfaced from the Stinger initiative including the use of the ORC file format and Tez as the execution engine

Demo: Data Manipulation with Hive

Lesson Objectives

After completing this lesson, students should be able to:

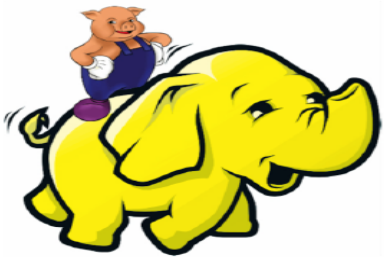
- Describe how Pig works
  - List the execution approaches available
  - Use basic commands and available libraries
  - Explain how Hive's HCatalog allows Pig to leverage defined schemas
- Observe the demonstration: *Risk Analysis with Pig*

- Pig Overview

Objectives

Hadoop Ecosystem – Pig

- An engine for executing programs on top of Hadoop
- It provides a language, Pig Latin, to specify these programs



Pig Latin

- High-level data-flow scripting language
- Pig executes in a unique fashion:
  - During execution, each statement is processed by the Pig interpreter
  - If a statement is valid, it gets added to a logical plan built by the interpreter
  - The steps in the logical plan do not actually execute until a DUMP or STORE command is used

Why Use Pig?

- **Example Scenario:**
  We want to target a subset of our users and then determine the most popular pages they access

```
users = LOAD 'input/users' USING PigStorage(',')
        AS (name:chararray, age:int);

filtrd = FILTER users BY age >= 18 and age <= 25;

pages = LOAD 'input/pages' USING PigStorage(',')
        AS (user:chararray, url:chararray);

jnd = JOIN filtrd BY name, pages BY user;

grpd = GROUP jnd BY url;

smmd = FOREACH grpd GENERATE group, COUNT(jnd) AS clicks;

srtd = ORDER smmd BY clicks DESC;

top5 = LIMIT srtd 5;

STORE Top5 INTO 'output/top5sites' USING PigStorage(',');
```

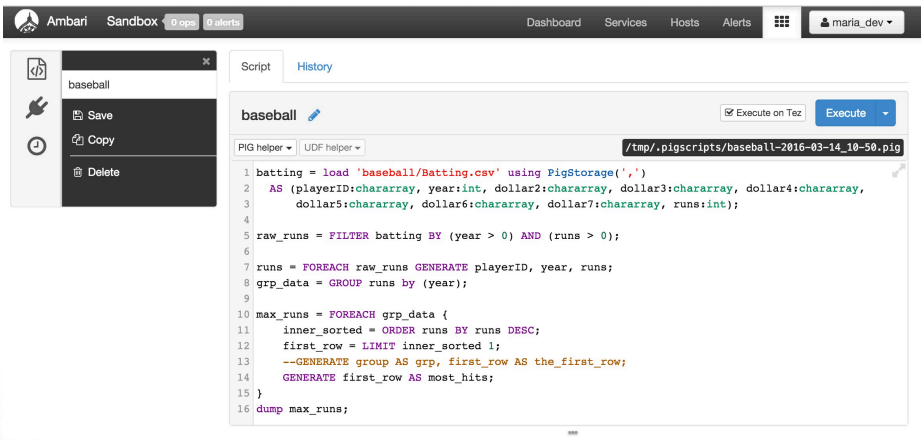
The Grunt Shell

- An interactive shell for entering Pig Latin statements
- Started by running the pig executable

Grunt shell

```
rich — root@sandbox:~ — ssh — 59×5
grunt> employees = LOAD 'pigdemo.txt' AS (state, name);
grunt> describe employees;
employees: {state: bytearray,name: bytearray}
grunt> employees_grp = group employees by state;
grunt> dump employees;
```


Executing Scripts in Ambari Pig View



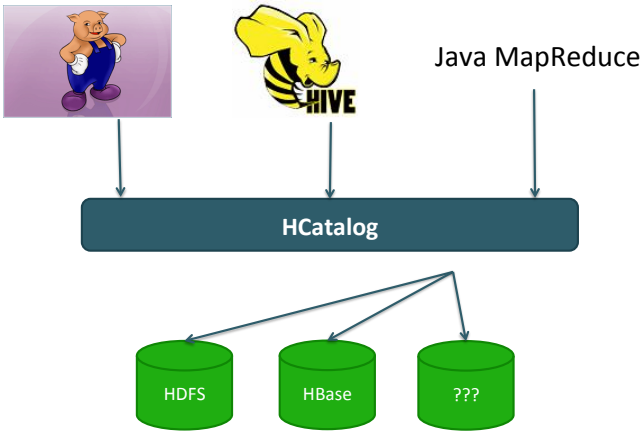
Simple and Novel Commands

Pig Command	Description
LOAD	Read data from file system
STORE	Write data to file system
FOREACH	Apply expression to each record and output 1+ records
FILTER	Apply predicate and remove records that do not return true
GROUP/COGROUP	Collect records with the same key from one or more inputs
JOIN	Joint 2+ inputs based on a key; various join algorithms exist
ORDER	Sort records based on a key
DISTINCT	Remove duplicate records
UNION	Merge two data sets
SPLIT	Split data into 2+ more sets based on filter conditions
STREAM	Send all records through a user provided executable
SAMPLE	Read a random sample of the data
LIMIT	Limit the number of records

Overview of the DataFu Library

- A collection of Pig UDFs for data analysis on Hadoop
- Started by LinkedIn and open-sourced under the Apache 2.0 license
- Includes functions for:
  - Bag and set operations
  - PageRank
  - Quantiles
  - Variance
  - Sessionization

HCatalog in the Ecosystem



Knowledge Check

Questions

1. What is the name of the programming language that Pig uses?
2. What are the two primary ways to execute Pig?
3. What two commands trigger a logical plan to begin execution?
4. List the name of the popular data analysis library presented in this module.
5. What Hive component allows Pig to take advantage of predefined attribute names and data types?

Summary

Summary

- Pig is a high-level data-flow scripting language
- Scripts do not execute until an I/O operation like DUMP or STORE are reached
- Can be run via the interactive shell or as a script
- Has a comprehensive set of commands available to Pig programmers
- DataFu library is a collection of Pig UDFs for data analysis on Hadoop
- HCatalog provides a consistent data model for the various tools that use Hadoop

Demo: Risk Analysis with Pig

Lesson Objectives

After completing this lesson, students should be able to:

- Describe Spark with special focus on
  - RDD definition
  - Extensions to Spark Core
- Discuss performance considerations
  - In-memory benefits & options
  - Task scheduling and execution
  - Fine-grained controls on parallelization
- List additional frameworks that layer on top of Spark Core
  - SQL, Streaming & MLlib
  - Apache Zeppelin
- Observe the demonstration: *Risk Analysis with Spark*

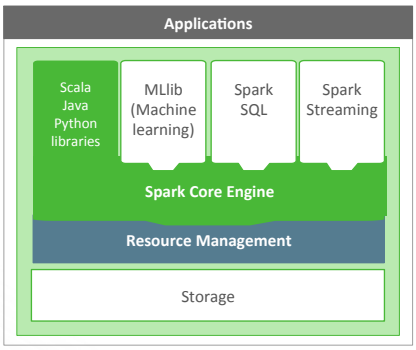
Objectives

- Apache Spark Overview

What is Spark?

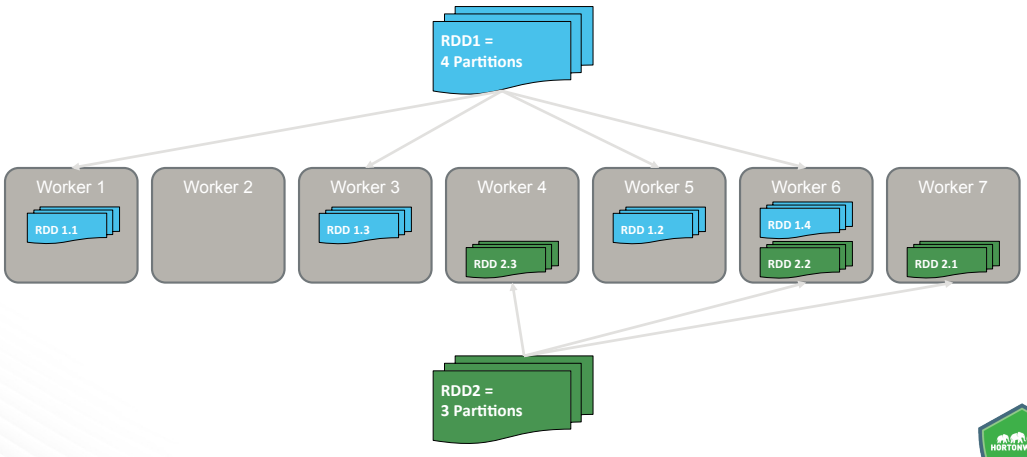
- An open-source software solution that performs rapid calculations on in-memory datasets
- Open Source [Apache hosted & licensed]
  - Free to download and use in production
  - Developed by a community of developers
- Supports use of well-known languages such as: Scala, Python, R, Java
- Datasets - RDD
  - **RDD** (Resilient Distributed Dataset) is the basis for what Spark enables
  - **Resilient**: the models can be recreated on the fly from known state
  - **Distributed**: the dataset is often partitioned across multiple nodes for increased scalability and parallelism

Apache Spark



- A data access engine for fast, large-scale data processing
- Designed for iterative in-memory computations and interactive data mining
- Provides expressive multi-language APIs for Scala, Java and Python
- Data workers can use built-in libraries to rapidly iterate over data for:
  - ETL
  - Machine learning
  - SQL workloads
  - Stream processing

Cluster with Two RDDs



Spark Focus

Leverage Data in HDP	• Efficient HBase connector to push predicates and prune queries backed by HBase • Hive as Spark Data source • Improve ORC Data Source efficiency
Improve multi tenancy	• HDFS Memory Tier to provide low latency cross context access • REST API for Spark job management • Spark Thrift Server security enhancements
Spark runs best on YARN	• Dynamic executor allocation uses cluster resources on demand • More efficient cluster utilization with YARN container resize • Token renewal for long running Spark jobs • Leverage GPUs for Spark Jobs on YARN

Spark Shell vs. Spark Applications

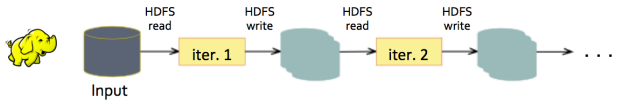
- **Spark shell** allows interactive manipulation/exploration of data
  - Data discovery
  - Building pipelines interactively
- **Spark applications** run as independent programs
  - ETL Processing
  - Streaming
  - Model Building
- Standalone applications can be scheduled

Objectives

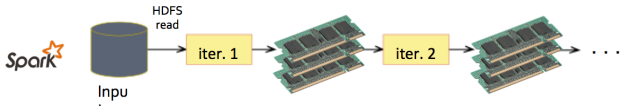
- Apache Spark Overview
- Performance Considerations

Spark Motivation

- **MapReduce** – involves *lots of disk I/O* (Disk I/O is very slow)



- **Spark** – Keep more data in memory



Spark RDD Persist Options

Storage Level	Where?	Storage format	Comments
MEMORY_ONLY	RAM	Deserialized	Default level
MEMORY_AND_DISK	RAM and DISK	Deserialized	Disk is backup for partitions that don't fit in memory
MEMORY_ONLY_SER	RAM	Serialized	Reduced RAM but more CPU intensive
MEMORY_AND_DISK_SER	RAM AND DISK	Serialized	Reduced RAM but more CPU intensive
DISK_ONLY	DISK	Deserialized	
MEMORY_ONLY_2	RAM	Deserialized	Stores each partition on two cluster nodes
MEMORY_AND_DISK_2	RAM AND DISK	Deserialized	Stores each partition on two cluster nodes
OFF_HEAP	TACHYON	Serialized	Experimental

Spark vs. MapReduce (1 of 2)

- Higher level API
- In-memory data storage
  - Up to 100x performance improvement

Pyspark

```
text_file = spark.textFile("hdfs://...")
counts = text_file.flatMap(lambda line: line.split(" ")) \
             .map(lambda word: (word, 1)) \
             .reduceByKey(lambda a, b: a + b)
counts.saveAsTextFile("hdfs://...")
```

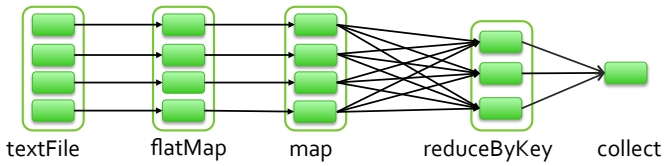
Java MapReduce

Spark vs. MapReduce (2 of 2)

- **Why is Spark faster?**
  - Caching data to memory can avoid extra reads from disk
  - Scheduling of tasks from 15-20s to 15-20ms
  - Resources are dedicated the entire life of the application
  - Can link multiple maps and reduces together without having to write intermediate data to HDFS
  - Every reduce doesn't require a map

Still Based on MapReduce Principles

```
sc.textFile("/some-hdfs-data") \                RDD[String]
 .flatMap(lambda line: line.split(" ")) \       RDD[List[String]]
 .map(lambda line: (word, 1))) \                RDD[(String, Int)]
 .reduceByKey(lambda a,b : a+b, \               RDD[(String, Int)]
       numPartition=3) \
 .collect()                                     Array[(String, Int)]
```



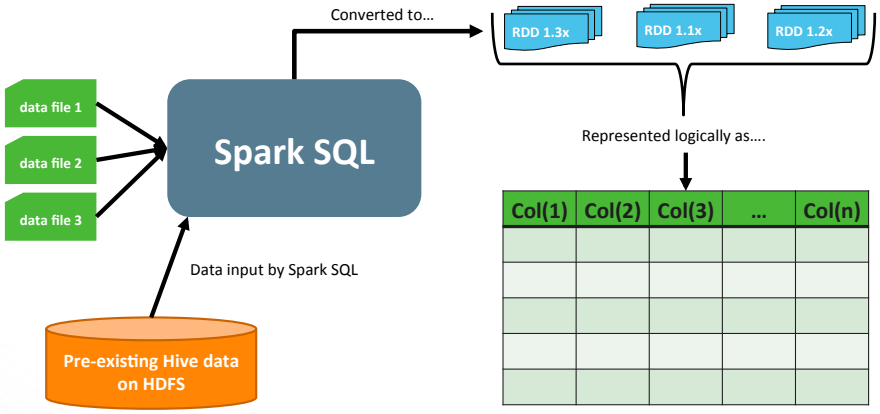
Objectives

- Apache Spark Overview
- Performance Considerations
- Beyond Spark Core

Spark SQL Overview

- A module built on top of Spark Core
- Provides a programming abstraction for distributed processing of large-scale structured data in Spark
- Data is described as a DataFrame with rows, columns and a schema
- Data manipulation and access is available with two mechanisms
  - SQL Queries
  - DataFrames API

The DataFrame Visually

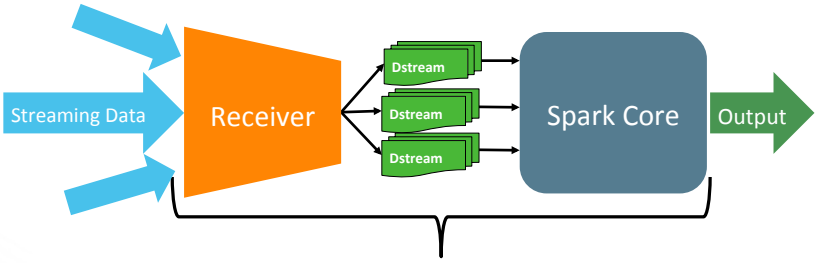


Spark Streaming Overview

- **Extension of Spark Core**
  - Library built on top of Spark Core
  - Reuses a lot of the same APIs
- **Utilizes a micro-batch architecture**
  - Process batches of data, as small as 1s latency
- **Process batches of data called Dstreams**
  - Discretized Streams of data
  - Share very little physically with RDD's, but are used similar to RDD's

Architecture Overview

- Streaming Applications consist of the same components as a Core application, but add the concept of a receiver
- The receiver is a process running on an executor

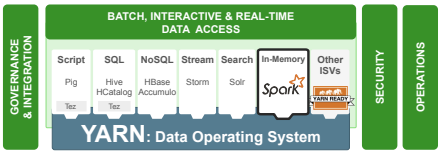


MLlib Overview

- A Spark implementation of common machine learning algorithms and utilities
- Includes:
  - Classification
  - Regression
  - Clustering
  - Collaborative filtering
  - Dimensionality reduction
- MLlib allows data scientists the ability to easily scale machine learning algorithms on a Hadoop cluster.

Hortonworks Commitment to Spark

Hortonworks is focused on making Apache Spark enterprise-ready, so you can depend on it for mission critical applications



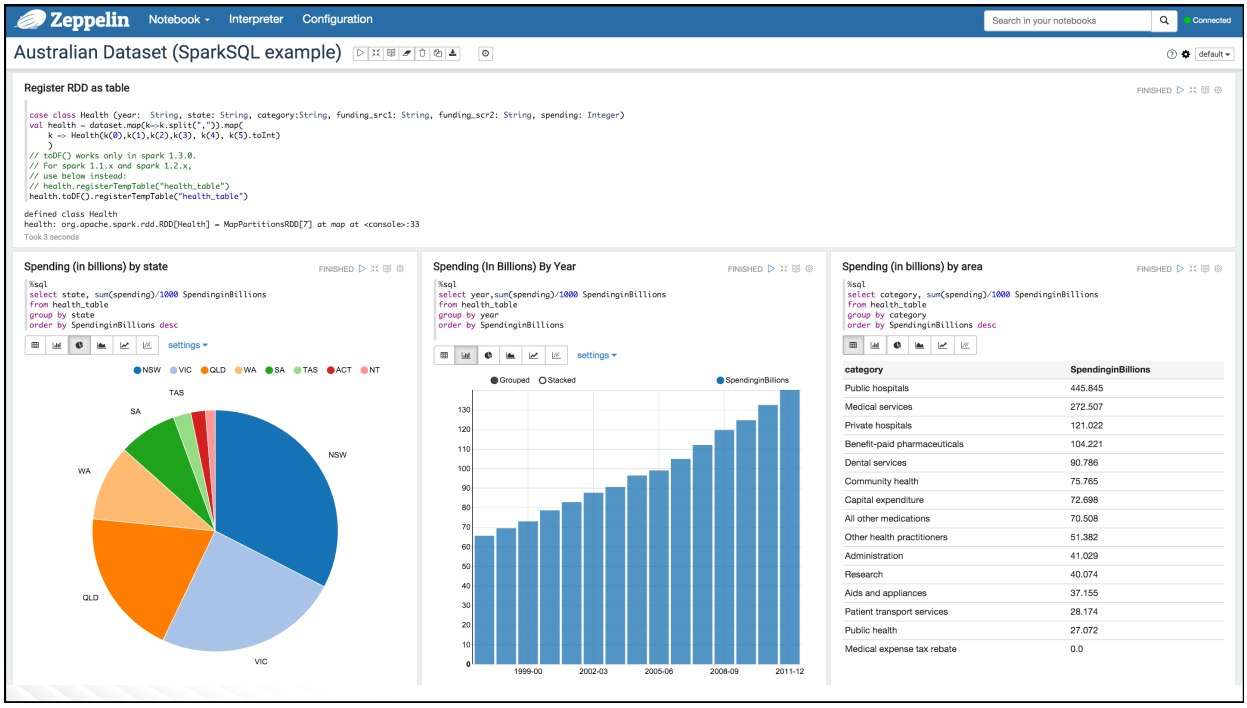
1. **YARN enables Spark to co-exist with other engines**
   Spark is "YARN Ready" so its memory & CPU intensive apps can work with predictable performance alongside other engines all on the same sets of data.

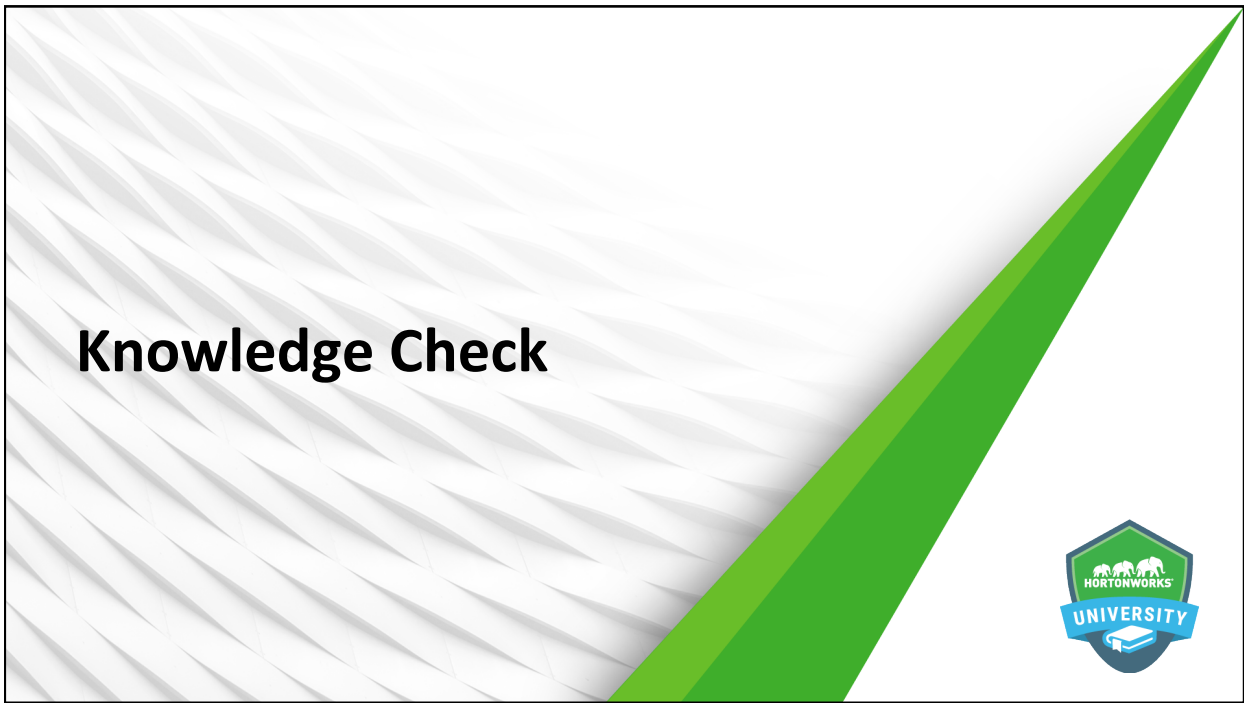
2. **Extend Spark with enterprise capabilities**
   Ensure Spark can be managed, secured and governed all via a single set of frameworks to ensure consistency. Ensure reliability and quality of service of Spark along side other engines.

3. **Actively collaborate within the open community**
   As with everything we do at Hortonworks, we work entirely within the open community across Spark and all related projects to improve this key Hadoop technology.

Apache Zeppelin

Features	Use Cases
A web-based notebook for interactive analytics —Ad-hoc experimentation with Spark, Hive, Shell, Flink, Tajo, Ignite, Lens, etc. Deeply integrated with Spark and Hadoop —Can be managed via Ambari Stacks Supports multiple language backends —Pluggable "Interpreters" Incubating at Apache —100% open source and open community	Data exploration and discovery Visualization—tables, graphs and charts Interactive snippet-at-a-time experience Collaboration and publishing "Modern Data Science Studio"


Zeppelin
Notebook
Interpreter
Configuration
Search in your notebooks
Connected
Australian Dataset (SparkSQL example)
default
Register RDD as table
FINISHED
case class Health (year: String, state: String, category:String, funding_src1: String, funding_scr2: String, spending: Integer)
val health = dataset.map(k=>k.split(",")).map(
k => Health(k(0),k(1),k(2),k(3), k(4), k(5).toInt)
)
// toDF() works only in spark 1.3.0.
// For spark 1.1.x and spark 1.2.x,
// use below instead:
// health.registerTempTable("health_table")
health.toDF().registerTempTable("health_table")
defined class Health
health: org.apache.spark.rdd.RDD[Health] = MapPartitionsRDD[7] at map at <console>:33
Took 3 seconds
Spending (in billions) by state
%sql
select state, sum(spending)/1000 SpendinginBillions
from health_table
group by state
order by SpendinginBillions desc
settings
NSW VIC QLD WA SA TAS ACT NT
TAS
SA
NSW
WA
QLD
VIC
Spending (In Billions) By Year
%sql
select year,sum(spending)/1000 SpendinginBillions
from health_table
group by year
order by SpendinginBillions
settings
Grouped
Stacked
SpendinginBillions
130
120
110
100
90
80
70
60
50
40
30
20
10
0
1999-00
2002-03
2005-06
2008-09
2011-12
Spending (in billions) by area
%sql
select category, sum(spending)/1000 SpendinginBillions
from health_table
group by category
order by SpendinginBillions desc
category
SpendinginBillions
Public hospitals
445.845
Medical services
272.507
Private hospitals
121.022
Benefit-paid pharmaceuticals
104.221
Dental services
90.786
Community health
75.765
Capital expenditure
72.698
All other medications
70.508
Other health practitioners
51.382
Administration
41.029
Research
40.074
Aids and appliances
37.155
Patient transport services
28.174
Public health
27.072
Medical expense tax rebate
0.0


Knowledge Check
HORTONWORKS
UNIVERSITY

Questions

1. What does RDD stand for?
2. List the languages supported by Spark.
3. What are some of the features that make up Spark's "secret sauce" for fast performance?
4. True/False? RDDs are not persisted in-memory by default.
5. Which Spark extension utilizes Dstreams? Which one introduced DataFrames?

Summary

Summary

- Spark houses data in an RDD structure and allows re-parallelization as needed
- The “sweet spot” is iterative in-memory computations and interactive data modeling
- Python, Scala, Java and R are supported languages
- Provides data processing, ETL, machine learning, stream processing, SQL querying
- In-memory caching is not a default setting and there are many options to choose from
- Maintains dedicated resources and its task scheduler is lightning fast
- Spark SQL has a DataFrame API In addition to classical SQL querying
- Spark Streaming uses micro-batches that are much like RDDs loaded from disk
- MLlib allows data scientists the ability to easily scale machine learning algorithms
- Apache Zeppelin is considered the “Modern Data Science Studio”

Demo: Risk Analysis with Spark

Lesson Objectives

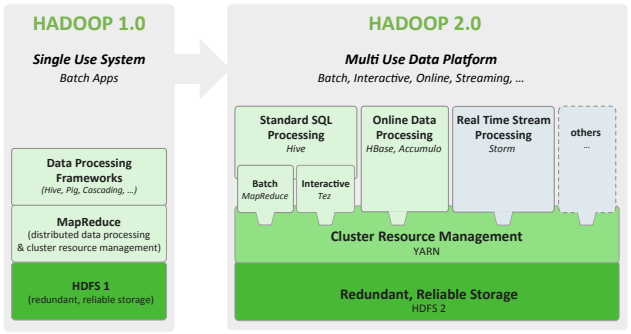
After completing this lesson, students should be able to:

- Describe the purpose and components of YARN
- Describe the major architectural components and their interactions
  - ResourceManager
  - NodeManager
  - ApplicationManager
- Describe additional YARN features
  - High Availability
  - Resource request model
  - Schedulers

Objectives

- YARN Overview

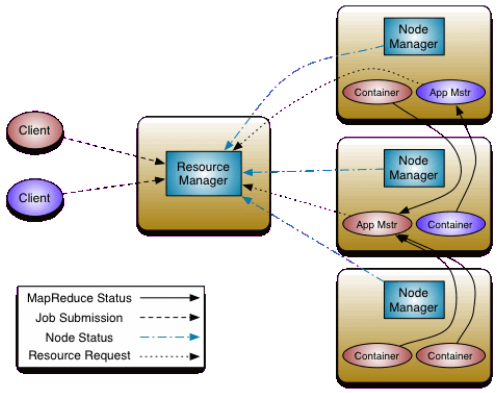
YARN Enables Multiple Workloads



Interact with all data in multiple ways simultaneously

YARN Architectural Components

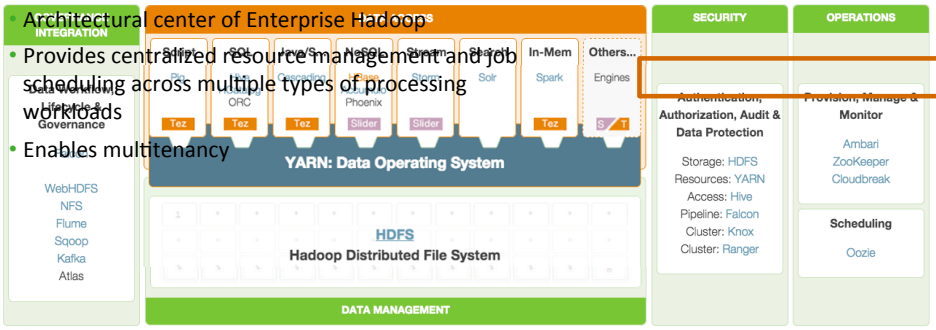
- **Resource Manager**
  - Global resource scheduler
  - Hierarchical queues
- **Node Manager**
  - Per-machine agent
  - Manages the life-cycle of container
  - Container resource monitoring
- **Application Master**
  - Per-application
  - Manages application scheduling and task execution
  - E.g. MapReduce Application Master



Objectives

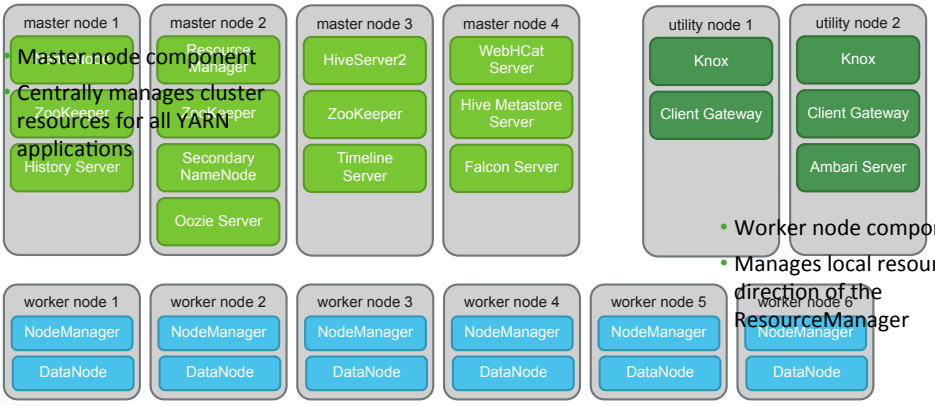
- YARN Overview
- YARN Components and Interactions

YARN Resource Management



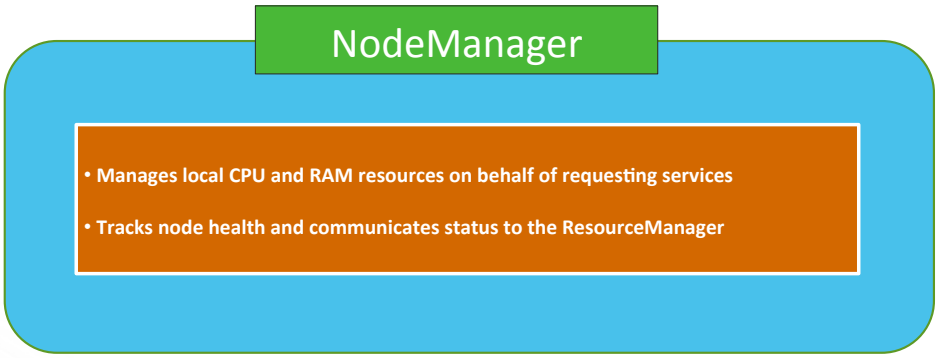
- Architectural center of Enterprise Hadoop
- Provides centralized resource management and job scheduling across multiple types of processing workloads
- Enables multitenancy

YARN Arcitecture – Big Picture View

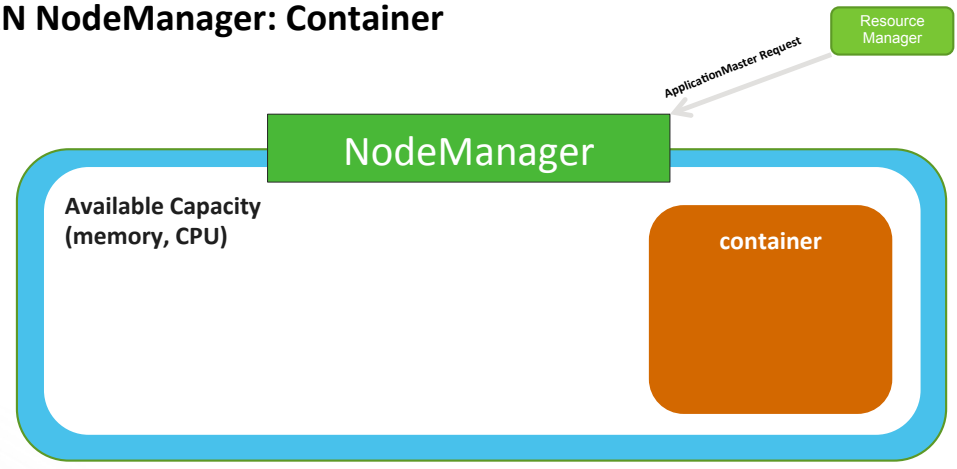


- Master node component
- Centrally manages cluster resources for all YARN applications
- Worker node component
- Manages local resources at the direction of the ResourceManager

YARN NodeManager



YARN NodeManager: Container

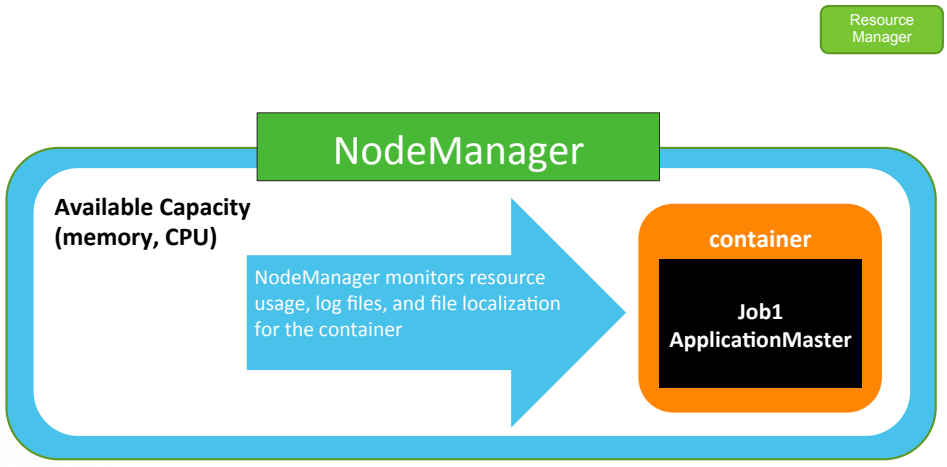


Containers Defined

container

- Allocated RAM and CPU cores by the NodeManager
- Runs ApplicationMaster job
- New container spawned for each discrete job task

YARN NodeManager – Application Master

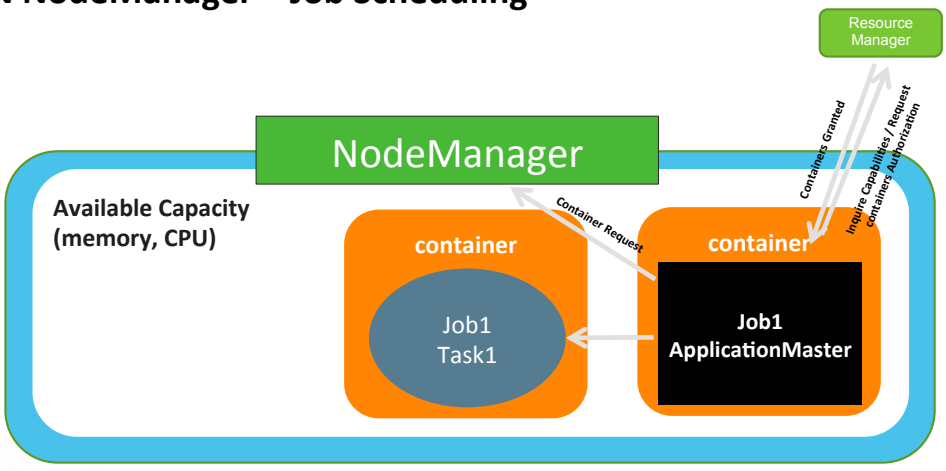


ApplicationMaster Defined

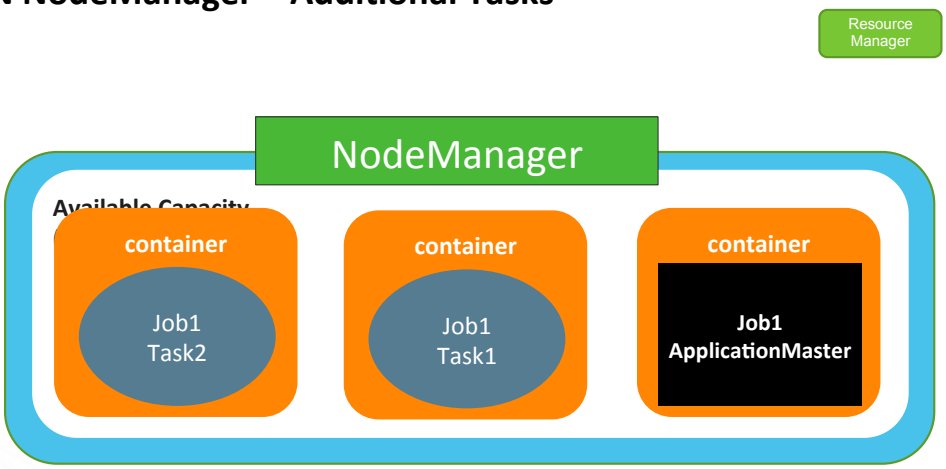
ApplicationMaster

- Bootstrap process for YARN applications
- Negotiates for resources with ResourceManager
- Works with NodeManagers to configure and execute containers and monitors application resource consumption
- Provides application fault tolerance and thus significant horizontal scale capabilities

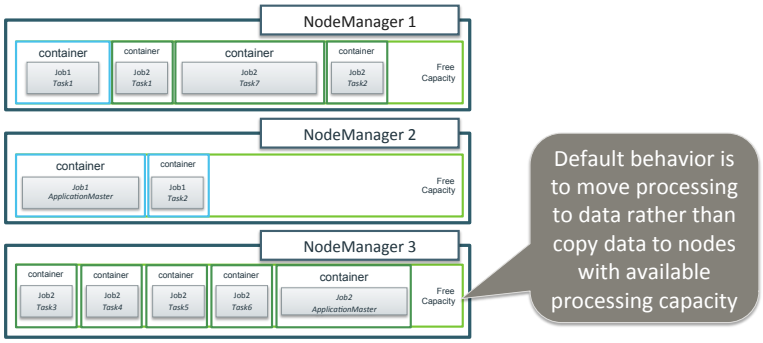
YARN NodeManager – Job Scheduling



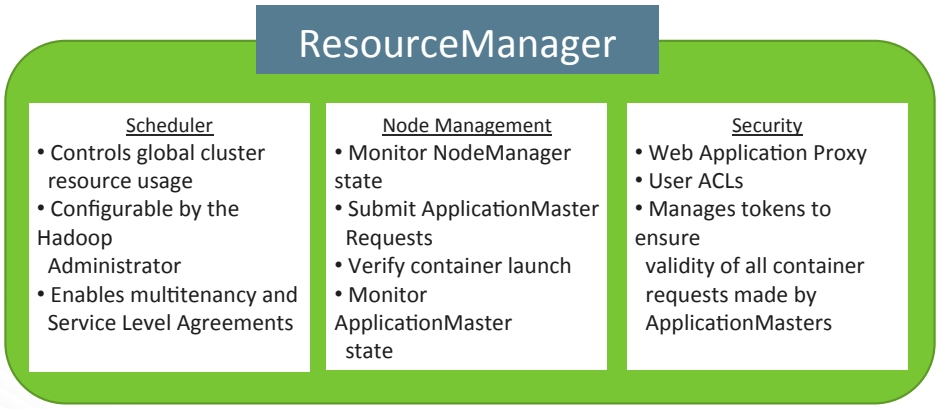
YARN NodeManager – Additional Tasks



Example: Multi-node Resource Allocation Scenario



YARN ResourceManager (Master Node)



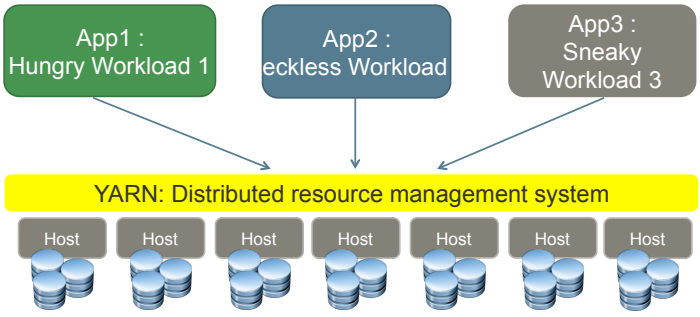
Objectives

- YARN Overview
- YARN Components and Interactions
- Additional YARN Features

ResourceManager High Availability

- **The YARN ResourceManager is a single point of failure.**
  - The entire cluster is unavailable if the ResourceManager:
  - Fails or becomes unreachable
  - Is stopped to perform maintenance
- **ResourceManager HA:**
  - Uses a redundant ResourceManager
  - Is configured in an Active/Standby configuration
  - Enables fast failover in response to ResourceManager failure
  - Permits administrator-initiated failover for maintenance
  - Is configured by Ambari

Shared Infrastructure Challenges



YARN containers provides required Isolation, Resource Management and Security

Resource Request

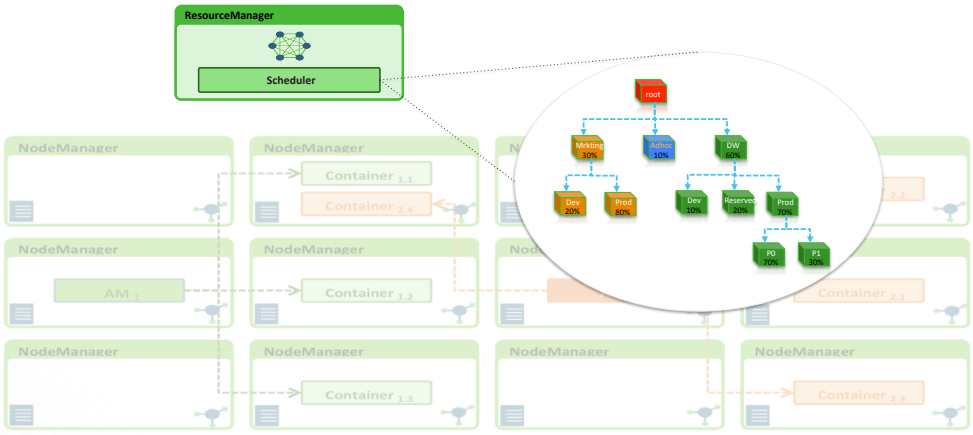
- Fine-grained resource ask to the ResourceManager
- Ask for a specific amount of resources (memory, CPU, etc.) on a specific machine or rack
- Use special value of * for resource name for any machine

ResourceRequest
priority
resourceName
capability
numContainers

Application Resource Request

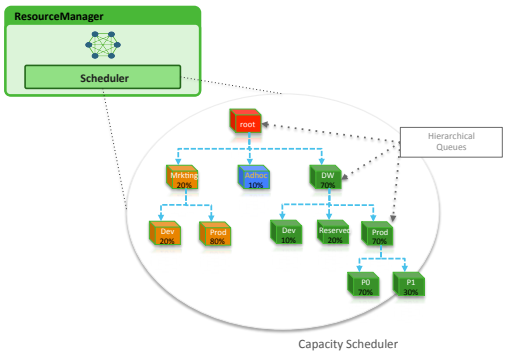
priority	capability	resourceName	numContainers
0	<2gb, 1 core>	host01	1
		rack0	1
		*	1
1	<4gb, 1 core>	*	1

Getting a Fair Share – Capacity Scheduler



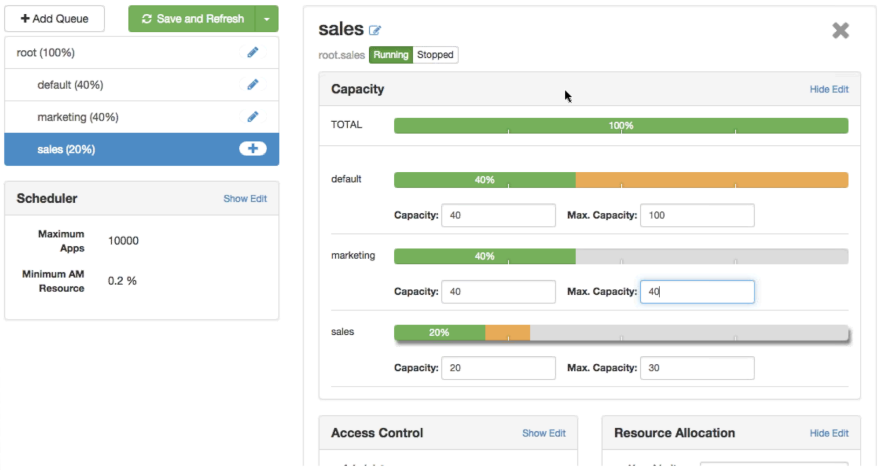
Multi-tenancy with Capacity Scheduler

- **Queues**
- **Economics as queue-capacity**
  - Hierarchical Queues
- **SLAs**
  - Preemption
- **Resource Isolation**
  - Linux: CGroups
  - MS Windows: Job Control
- **Administration**
  - Queue ACLs
  - Run-time re-configuration for queues
  - Charge-back



Capacity Scheduler

Managing Queue Limits with Ambari



Policy-based Use of Computing Resources

Scheduler Queues

- Capacity Scheduler allows for multiple tenants to share resources
- Queues limit access to resources
- Sub-queues are possible allowing capacity to be shared within a tenant
- Each queue has ACLs associated with users and groups
- Capacity guarantees can be set to provide minimum resource allocations
- Soft and hard limits can be placed on queues

Tuning queues and limits minimizes idle resources.

Hadoop Cluster

"Root" queues can be setup for each tenant, and sub-queues for logical division within tenants. Each queue is allocated a % of total capacity. Queues at the root level divide the actual resources.

Queues do not allocate specific resources, only a % of the total.

Sub-Queues sub-divide % of allocated resources.

Knowledge Check

Questions

1. The master node service is called the _____ and the _____ runs on the worker nodes.
2. Which Container resource type is the driver for most resource requests?
3. True/False? ApplicationMasters execute on master nodes.
4. What component is responsible for dealing with a Container failure?
5. True/False? Capacity Scheduler queues are aligned with specific worker nodes.

Summary

Summary

- YARN enables multiple workloads to execute simultaneously in the cluster
- The ResourceManager is the master process responsible for fulfilling resource requests and the NodeManager resides on the worker nodes along with the actual Containers that fulfill job functions
- The ApplicationMaster resides within a Container and is the process responsible for running a job (batch or long-lived service) and making appropriate resource requests
- The Capacity Scheduler allows for resource sharing that enables SLA-enabled multi-tenancy

Lesson Objectives

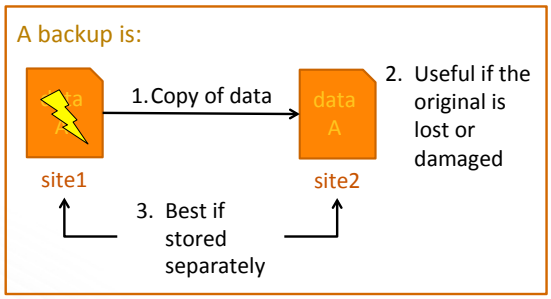
After completing this lesson, students should be able to:

- Identify the importance of Hadoop backups
  - Summarize Hadoop backup considerations
  - Describe the purpose HDFS snapshots
  - Explain the purpose of Distributed Copy (DistCp)
- Observe the demonstration: *Data Backup with Falcon* (Time Permitting)

Objectives

- Protecting a Cluster with Backups

The Importance of Backups



- Data is already protected by HDFS built-in redundancy? Why should it be backed up again?
- Data from multiple systems or entire data centers can become unavailable because of:
  - System failures
    - Hardware, software, power, cooling
  - Natural disasters
    - Fire, flood, hurricane, earthquake, …
  - Man-made disasters
    - Human error, criminal acts, terrorism, war…

What to Back Up?

When planning for backups, consider:

Application Data

- HDFS
  - Files and directories
  - Hive tables
  - HBase tables

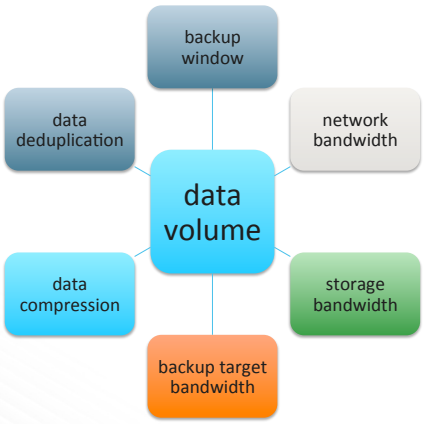
Hadoop Metadata

- NameNode metadata
- Databases
  - Ambari
  - Hive
  - Oozie
  - Ranger
- ZooKeeper znode data

Hadoop Configuration

- Operating system `/etc` subdirectories
- Ambari Server `/etc` configuration directories

Hadoop Backup Considerations



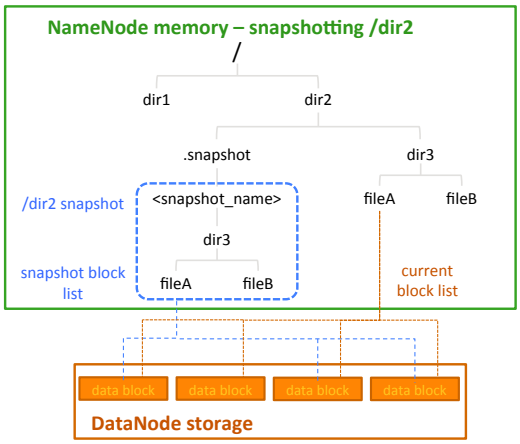
- Many factors are affected by the amount of data to back up:
  - Sufficient time for the backup window?
  - Sufficient network bandwidth?
  - Sufficient storage bandwidth?
  - Sufficient backup device bandwidth?
  - Is HDFS data compressed?
    - Compressed data unlikely to be further compressed for back up
  - Does backup support deduplication?
    - Only useful if file backed up more than once

HDFS Snapshots

- An HDFS snapshot is a point-in-time image of data.
- The snapshot is read-only.
- The snapshot can be for a directory or an entire file system.
- HDFS snapshots are useful for:
  - Protection against user error
  - Backups
  - Test and development
  - Disaster recovery

HDFS Snapshot Operation

- Snapshots are created in NameNode memory.
  - Fast to create, no data blocks copied
  - Only the NameNode is aware of snapshots
- DataNodes are unaware of snapshots
  - DataNodes only maintain data blocks
- NameNode block lists of current file and snapshot could be different
  - Current file could be appended with new data blocks
  - File could be deleted and exist only as a snapshot



Distributed Copy (DistCp)

- Copies HDFS data between or within Hadoop clusters
  - Copies a file or an entire directory
  - Can copy from multiple source directories to a single target directory
- Uses MapReduce mappers (no reducers) to perform the copy
  - MapReduce distributes the I/O load
  - Leverages MapReduce built-in error handling and reporting
- DistCp supports HDFS, S3, Cassandra, and more.
- DistCp supports copying between different HDFS versions.
- DistCp can create a significant load on a cluster.
- `hadoop distcp -help` to view command options

Backup using Snapshots and `DistCp`

Snapshot HDFS data to back up

```
hdfs dfs -createSnapshot /foo backup_snap
```

DistCp snapshotted files to backup cluster

```
hadoop distcp -update -p rbugp -strategy dynamic
hdfs://<namenode1>:8020/foo/.snapshot/backup_snap
hdfs://<namenode2>:8020/bar
```

Delete the HDFS snapshot

```
hdfs dfs -deleteSnapshot /foo backup_snap
```

Note: Apache Oozie has DistCp built-in to it as one of its functions. Apache Falcon can automate repetitive tasks using Oozie as its workflow engine. Falcon and Oozie can be configured to automate backups.

Knowledge Check

Questions

1. If HDFS data is already redundant across multiple DataNodes, why do you need to back it up?
2. True or false? Snapshots are fast because a minimal number of data blocks are copied when a snapshot is initially created.
3. Which two Apache frameworks can be used to automate HDFS backups?

Summary

Summary

- It is still important to back up Hadoop because of system failures, and man-made or natural disasters.
- Consider HDFS data, metadata, and Hadoop configuration files for backup.
- An HDFS snapshot is a read-only, point-in-time image of data.
- Snapshot creation is fast because a snapshot is created in NameNode memory; no data blocks are copied.
- DistCp copies HDFS data between or within Hadoop clusters.
- HDFS snapshots can be combined with DistCp to back up HDFS data to backup clusters.

Demo: Data Backups with Falcon

Lesson Objectives

After completing this lesson, students should be able to discuss:

- Explain how HDP addresses the key security requirements
  - Authentication & Authorization
  - Audit & Administration
  - Data Protection
- Visualize a typical multi-layered deployment strategy for security
  - Implementing Kerberos
  - Enhancing Authentication & Audit with Ranger
  - Securing the perimeter with Knox
  - Encrypt data at-rest and in-motion
- Explain the benefits and high-level architecture of Apache Ranger
- Observe the demonstration: *Securing Hive with Ranger* (Time Permitting)

Objectives

- HDP Security Solution Overview

The Security Challenge

- In order to protect any data system you must implement the following

Administration
Centrally management & consistent security

Authentication
Authenticate users and systems

Authorization
Provision access to data

Audit
Maintain a record of data access

Data Protection
Protect data at rest and in motion

3 Reasons for Security Focus

- **Malicious intent**
- **Unintentional breach**
- **Compliance**

HDP Security: Comprehensive, Complete and Simple

- Security in HDP is the most comprehensive and complete for Hadoop

Administration
Central management & consistent security

Authentication
Authenticate users and systems

Authorization
Provision access to data

Audit
Maintain a record of data access

Data Protection
Protect data at rest and in motion

- **HDP** ensures **comprehensive** enforcement of security policy across the entire Hadoop stack
- **HDP** provides functionality across the **complete** set of security requirements
- **HDP** is the only solution to provide a single **simple** interface for security policy definition and maintenance

Security Today in Hadoop with HDP

HDP

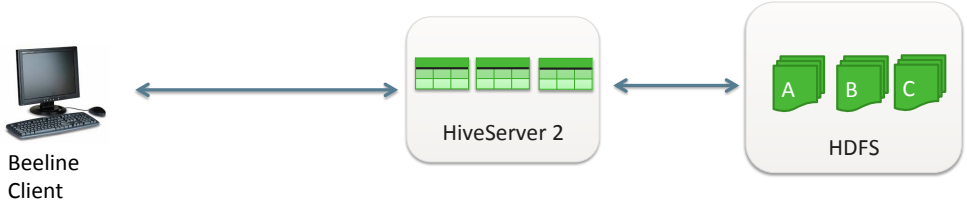
Centralized Security Administration w/ Ranger

Authentication Who am I/prove it?	**Authorization** What can I do?	**Audit** What did I do?	**Data Protection** Can data be encrypted at rest and over the wire?
• *Kerberos* • API security with *Apache Knox*	• Fine grain access control with *Apache Ranger*	• Centralized audit reporting w/ *Apache Ranger*	• Wire encryption in Hadoop • *Native* and *partner* encryption

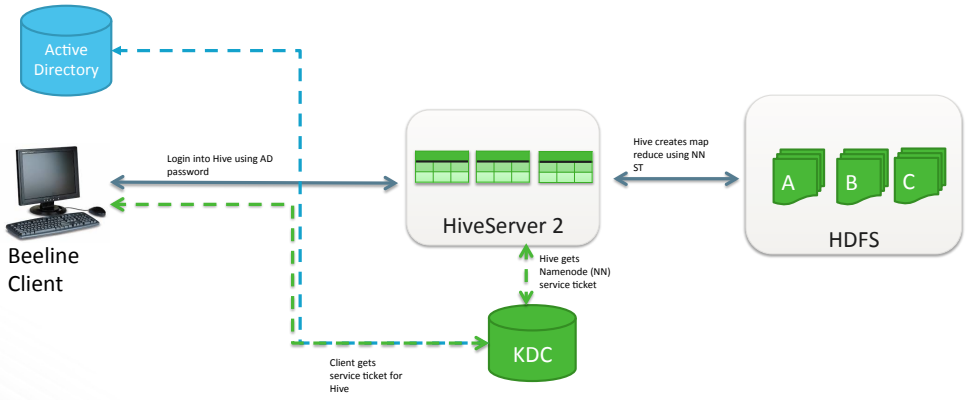
Objectives

- HDP Security Solution Overview
- Multi-layered Protection

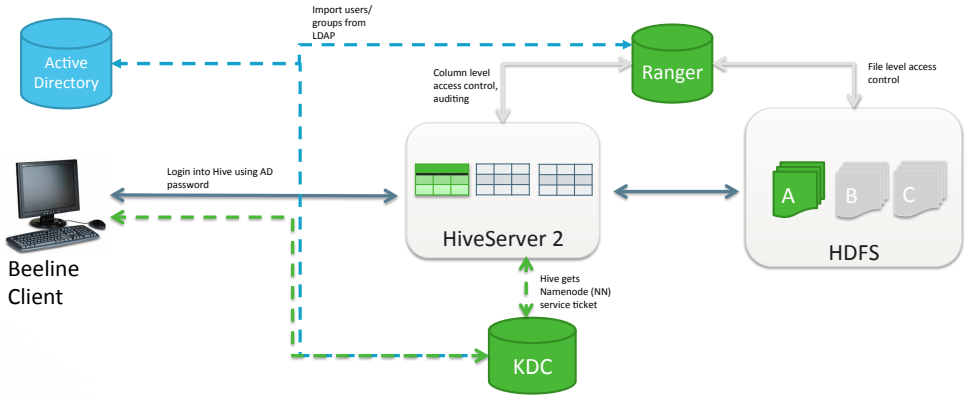
Typical Flow – SQL Access through Beeline Client



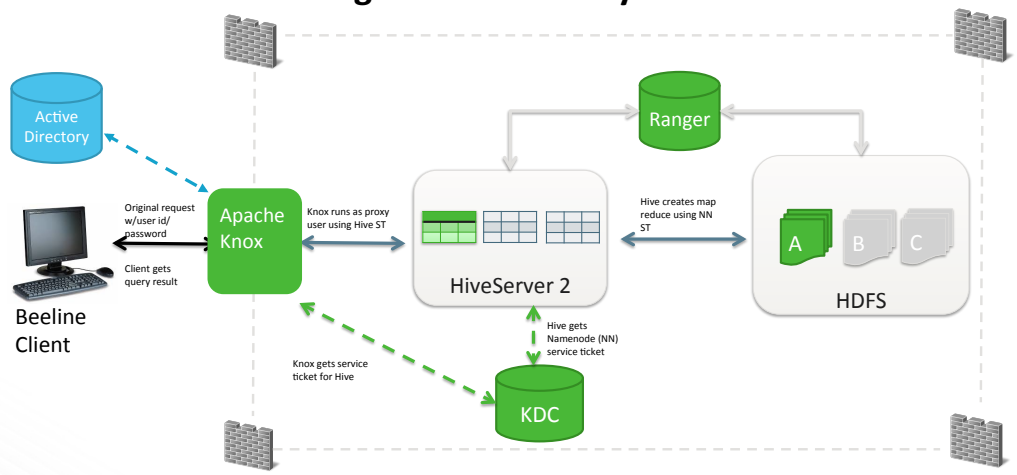
LAuthenticate through Kerberos



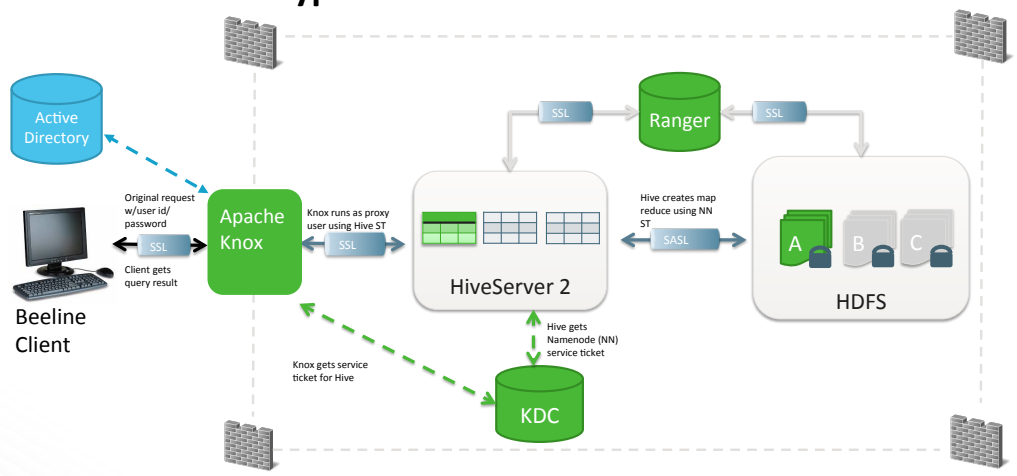
Add Authorization through Kerberos



Add Firewall Routed through Knox Gateway



Add Wire and File Encryption



Objectives

- HDP Security Solution Overview
- Multi-layered Protection
- Authorization and Auditing with Apache Ranger

Authorization and Audit

- **Authorization**
  Fine grain access control
  - HDFS – Folder, File
  - Hive – Database, Table, Column
  - HBase – Table, Column Family, Column
  - Storm, Knox and more

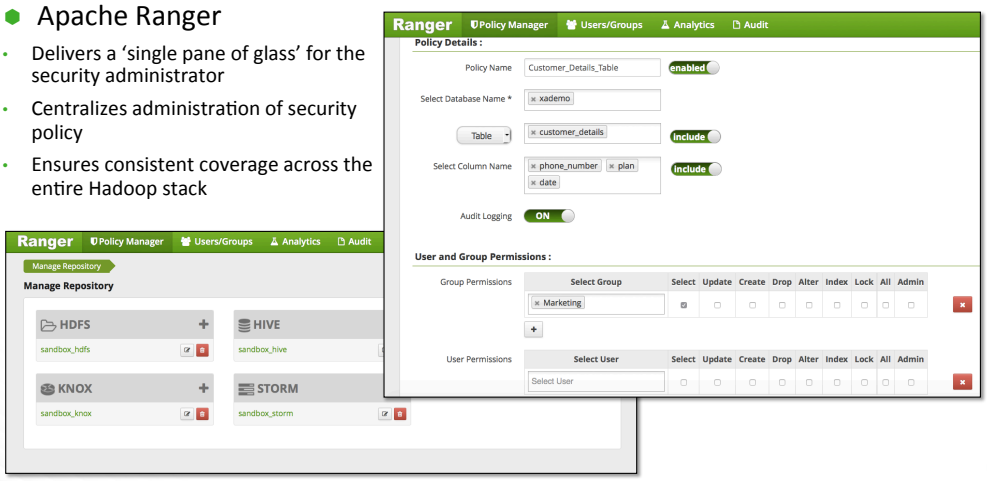
Flexibility in defining policies

- **Audit**
  Extensive user access auditing in HDFS, Hive and HBase
  - IP Address
  - Resource type/ resource
  - Timestamp
  - Access granted or denied

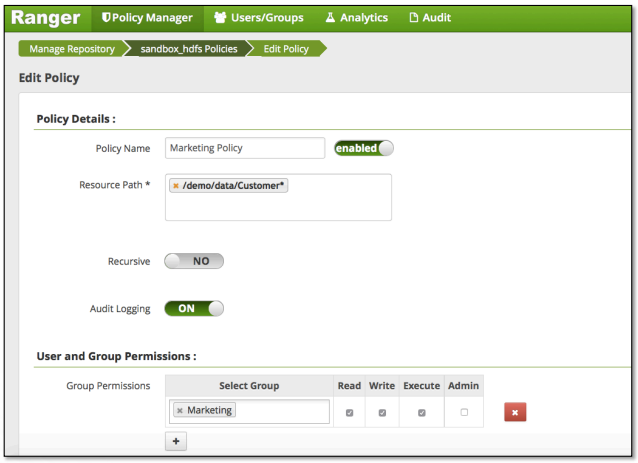
Control access into system

Central Security Administration

- Apache Ranger
  - Delivers a 'single pane of glass' for the security administrator
  - Centralizes administration of security policy
  - Ensures consistent coverage across the entire Hadoop stack



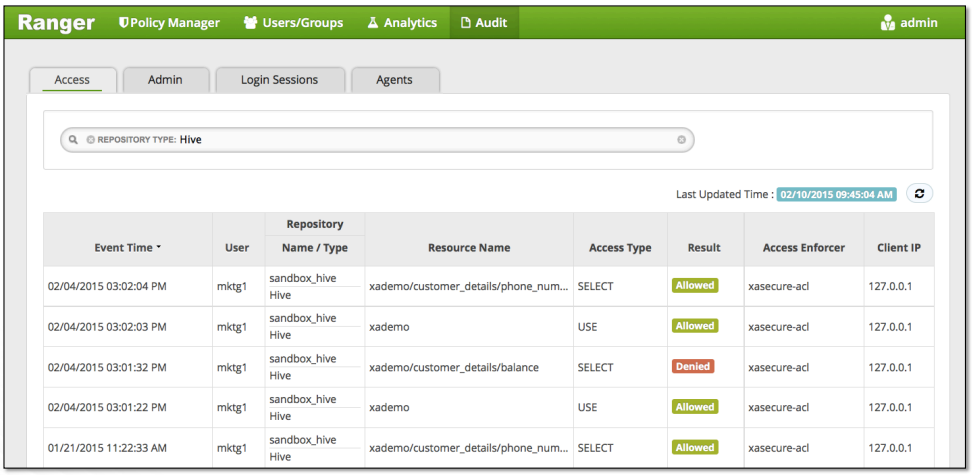
Set Up Authorization Policies



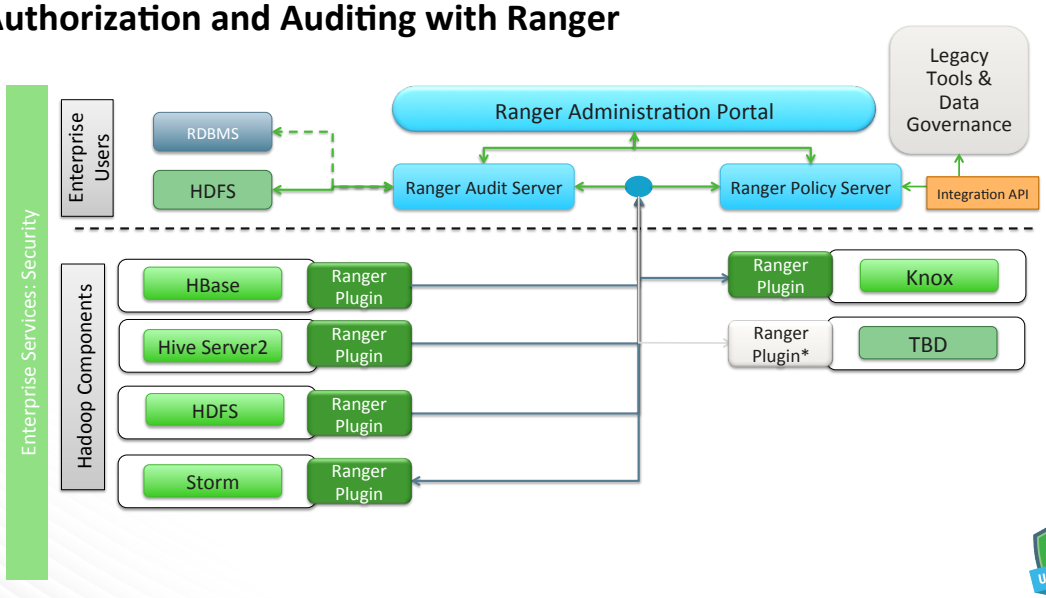
file level access control, flexible definition

Control permissions

Monitor through Auditing



Authorization and Auditing with Ranger



Knowledge Check

Questions

1. What are the primary requirements for security?
2. What technology guarantees strong authentication in Hadoop?
3. Which requirements does Ranger address?
4. Does Hadoop tackle data encryption at-rest or in-motion?

Summary

Summary

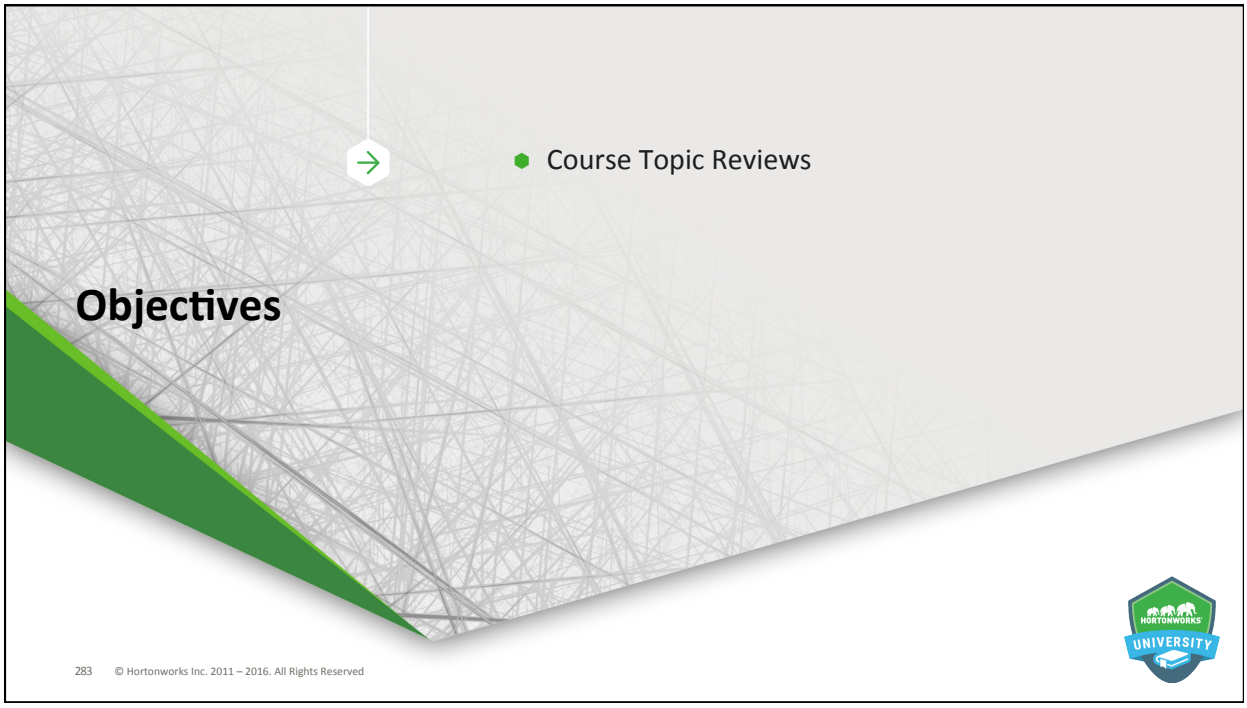
- HDP ensures comprehensive enforcement of security requirements across the entire Hadoop stack.
- Kerberos is the key to strong authentication.
- Ranger provides a single simple interface for security policy definition and maintenance.
- Encryption options available for data at-rest and in-motion.

Demo: Securing Hive with Ranger

Lesson Objectives

After completing this lesson, students should be able to discuss:

- Provide a brief review of the lesson topics in this course
- Describe possible next steps for additional training



The Case for Hadoop

- **The 3V's of Big Data are driving the adoption of Apache Hadoop (44 ZB by 2020)**
- **Existing data architectures make data inaccessible, incomplete, irrelevant, and expensive**
- **Hadoop is a scalable, fault tolerant, open source framework for the distributed storing and processing of large sets of data on commodity hardware**
- Six common use case families have emerged
  - Data Discovery
  - Single View
  - Predictive Analytics
  - Active Archive
  - ETL Offload
  - Data Enrichment
- YARN-centralized HDP = Open Enterprise Hadoop

284

The Hadoop Ecosystem

- **The five architectural pillars of Hadoop are:**
  - Data Management Frameworks
  - Data Access Frameworks
  - Governance & Integration Frameworks
  - Security Frameworks
  - Operations Frameworks
- **Primary server stereotypes are:**
  - Master nodes
  - Worker nodes
- **Hadoop complements existing systems and is the foundation of Connected Data Platforms**

HDFS Architecture

- **HDFS breaks files into blocks and replicates them for reliability and processing data locality**
- **The primary components are the master NameNode service and the worker DataNode service**
- The NameNode is a memory-based service
- The NameNode automatically takes care of recovery missing and corrupted blocks
- Clients interact with the NameNode to get a list, for each block, of DataNodes to write data to

Ingesting Data

- **There are many different ways to ingest data including customer solutions written via HDFS APIs as well as vendor connectors**
- **Streaming and batch workflows can work together in a holistic system**
- The NFS Gateway may help some legacy systems populate data into HDFS
- Sqoop's configurable number of database connection can overload an RDBMS
- The following are streaming frameworks:
  - Flume
  - Storm
  - Spark Streaming
  - HDF / NiFi

Parallel Processing Fundamentals

- **MapReduce is the foundational framework for processing data at scale because of its ability to break a large problem into any smaller ones**
- Mappers read data in the form of KVPs and each call to a Mapper is for a single KVP; it can return 0..m KVPs
- The framework shuffles & sorts the Mappers' outputted KVPs with the guarantee that only one Reducer will be asked to process a given Key's data
- Reducers are given a list of Values for a specific Key; they can return 0..m KVPs
- **Due to the fine-grained nature of the framework, many use cases are better suited for higher-order tools**

Apache Hive Overview

- **Hive is the data warehouse system for Hadoop and uses the familiar table and SQL metaphors that are used with classic RDBMS solutions**
- **The MetaStore maintains the logical view of tables as well as the physical characteristics such as where the data is stored and in what format it is in**
- Clients, using xDBC, connect to the HiveServer2 component on a master node which in turn submits queries into the worker nodes for processing
- Hive can create, populate and query tables
- Views are supported, but they are not materialized
- **Significant performance improvements have surfaced from the Stinger initiative including the use of the ORC file format and Tez as the execution engine**

Apache Pig Overview

- **Pig is a high-level data-flow scripting language**
- Scripts do not execute until an I/O operation like DUMP or STORE are reached
- Can be run via the interactive shell or as a script
- Has over 30 commands available to Pig programmers
- DataFu library is a collection of Pig UDFs for data analysis on Hadoop
- **HCatalog provides a consistent data model for the various tools that use Hadoop**

Apache Spark

- **Spark houses data in an RDD structure and allows re-parallelization as needed**
- **The "sweet spot" is iterative in-memory computations and interactive data modeling**
- Python, Scala, Java and R are supported languages
- **Provides data processing, ETL, machine learning, stream processing, SQL querying**
- In-memory caching is not a default setting and there are many options to choose from
- Maintains dedicated resources and its task scheduler is lightning fast
- Spark SQL has a DataFrame API In addition to classical SQL querying
- Spark Streaming uses micro-batches that are much like RDDs loaded from disk
- MLlib allows data scientists the ability to easily scale machine learning algorithms
- **Apache Zeppelin is considered the "Modern Data Science Studio"**

YARN Architecture

- **YARN enables multiple workloads to execute simultaneously in the cluster**
- The ResourceManager is the master process responsible for fulfilling resource requests and the NodeManager resides on the worker nodes along with the actual Containers that fulfill job functions
- The ApplicationMaster resides within a Container and is the process responsible for running a job (batch or long-lived service) and making appropriate resource requests
- **The Capacity Scheduler allows for resource sharing that enables SLA-enabled multi-tenancy**

Backup and Recovery

- **It is still important to back up Hadoop because of system failures, and man-made or natural disasters.**
- Consider HDFS data, metadata, and Hadoop configuration files for backup.
- **An HDFS snapshot is a read-only, point-in-time image of data.**
- Snapshot creation is fast because a snapshot is created in NameNode memory; no data blocks are copied.
- **DistCp copies HDFS data between or within Hadoop clusters.**
- HDFS snapshots can be combined with DistCp to back up HDFS data to backup clusters.

Hadoop Security

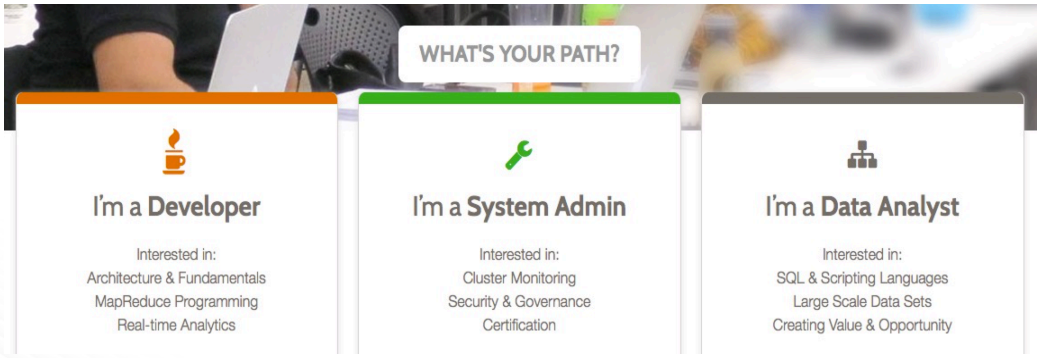
- **HDP ensures comprehensive enforcement of security requirements across the entire Hadoop stack**.
- Kerberos is the key to strong authentication.
- Ranger provides a single simple interface for security policy definition and maintenance.
- Encryption options available for data at-rest and in-motion.

Objectives

- Course Topic Summaries
- Next Steps for Training

Pick a Learning Path – http://hortonworks.com/training/

- Hortonworks University offers three distinct personas and has curricula for each:



Pick a Course – http://hortonworks.com/training/

Developer Courses	Operations Courses	Analyst Courses
Apache Pig and Hive	Hadoop Administration 1	Data Science
Java	Hadoop Administration 2	Apache HBase Essentials
Storm and Trident Fundamentals	Apache HBase Advanced Management	
Custom YARN Applications		
COMING Q2 2016		
Apache Spark - Python	Hortonworks Data Flow	
Apache Spark - Scala	Security	

Pick a Certification – http://hortonworks.com/training/certification/

- **HDP Certified Developer (HDPCD)**: for Hadoop developers using frameworks like Pig, Hive, Sqoop and Flume.
- **HDP Certified Administrator (HDPCA)**:for administrators who deploy and manage Hadoop clusters.
- **HDP Certified Developer:Java (HDPCD:Java)**: for developers who design, develop and architect Hadoop-based solutions written in the Java programming language.

Classes Available Worldwide Through Our Partners

Study Options Worldwide

In combination with our partner providers, classes are often available in numerous locations across the world.

Private On-site Training

Hortonworks training in-house covers all of our basic coursework, and provides a more intimate setting for 6 or more students.
Contact us for more details

Learn from the company focused solely on Hadoop.

What Makes Us Different?

1. Our courses are designed by the **leaders and committers** of Hadoop
2. We provide an **immersive** experience in **real-world** scenarios
3. We prepare you to **be an expert** with highly valued, **fresh skills**
4. Our courses are available **near you**, or accessible **online**

Hortonworks University courses are designed by the leaders and committers of Apache Hadoop. We provide immersive, real-world experience in scenario-based training. Courses offer unmatched depth and expertise available in both the classroom or online from anywhere in the world. We prepare you to be an expert with highly valued skills and for Certification.